# **XBee Python Library Documentation**

Release 1.4.0

Digi International Inc.

## Getting Started

1	Requ	iremer	nts		3
2	Cont	ents			5
	2.1	Gettin	g Started		5
	2.2		_	ation	5
	2.3				6
	2.4				6
	2.5	_			6
	2.6	_	_		6
		2.6.1		rted with XBee Python library	6
			2.6.1.1	Install your software	6
			2.6.1.2	Configure your XBee modules	8
			2.6.1.3	Run your first XBee Python application	11
		2.6.2	XBee t	erminology	14
			2.6.2.1	RF modules	14
			2.6.2.2	XBee RF modules	14
			2.6.2.3	Radio firmware	14
			2.6.2.4	Radio communication protocols	14
			2.6.2.5	Radio module operating modes	15
			2.6.2.6	API frames	17
			2.6.2.7	AT settings or commands	18
		2.6.3	Work v	vith XBee classes	18
			2.6.3.1	Instantiate an XBee device	19
			2.6.3.2	Open the XBee device connection	20
			2.6.3.3	Close the XBee device connection	23
		2.6.4	Configu	ure the XBee device	24
			2.6.4.1	Read and set common parameters	24
			2.6.4.2	Read, set and execute other parameters	27
			2.6.4.3	Apply configuration changes	29
			2.6.4.4	Write configuration changes	30
			2.6.4.5	Reset the device	31
			2.6.4.6	Configure Wi-Fi settings	31
			2.6.4.7	Configure Bluetooth settings	33
		2.6.5	Discov	er the XBee network	35
			2.6.5.1	Discovery types	36
			2.6.5.2	Deep discovery	37

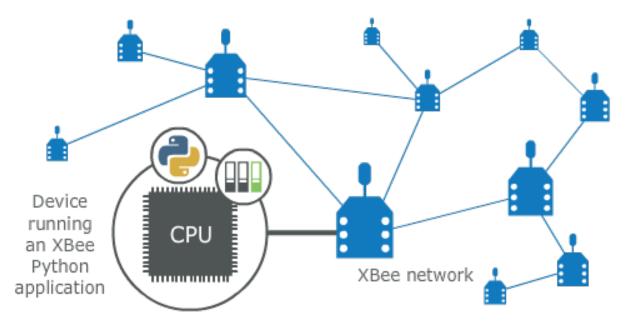
		2.6.5.3	Standard discovery	
		2.6.5.4	Discover the network	
		2.6.5.5	Access discovered nodes	
		2.6.5.6	Access connections between nodes	
		2.6.5.7	Add and remove nodes manually	
		2.6.5.8	Listen to network modification events	
	2.6.6		inicate with XBee devices	
		2.6.6.1	Send and receive data	
		2.6.6.2	Send and receive explicit data	
		2.6.6.3	Send and receive IP data	
		2.6.6.4	Send and receive SMS messages	
		2.6.6.5	Send and receive Bluetooth data	
		2.6.6.6	Send and receive MicroPython data	. 74
		2.6.6.7	Receive modem status events	. 76
		2.6.6.8	Communicate using XBee sockets	. 77
	2.6.7	Handle	analog and digital IO lines	. 81
		2.6.7.1	Configure the IO lines	
		2.6.7.2	Read IO samples	
		2.6.7.3	Change detection sampling	
	2.6.8	Update	the XBee	
		2.6.8.1	Update the XBee firmware	
		2.6.8.2	Update the XBee file system	
		2.6.8.3	Apply an XBee profile	
	2.6.9		ents	
	2.0.5	2.6.9.1	Logging level	
	2610		tython samples	
	2.0.10		Configuration samples	
			Network samples	
			Communication samples	
			IO samples	
			Firmware samples	
			File system samples	
			· · · · · · · · · · · · · · · · · · ·	
	2 6 11		Profile samples	
	2.0.11		ntly Asked Questions (FAQs)	
			What is XCTU and how do I download it?	
			How do I find the serial port and baud rate of my module?	
			Can I use the XBee Python Library with modules in AT operating mode?	
			I get the Python error ImportError: No module named 'serial'	
	0 ( 10		I get the Python error ImportError: No module named 'srp'	
	2.6.12	_	log	
			v1.4.0 - 03/18/2021	
			v1.3.0 - 11/05/2019	
			v1.2.0 - 04/05/2019	
			v1.1.1 - 04/25/2018	
			v1.1.0 - 01/19/2018	
			v1.0.0 - 10/02/2017	
	2.6.13		erence	
		2.6.13.1	digi package	. 124
3	Indices and	tables		1015
J	muices and	Lavies		1013
4	License			1017
Py	thon Module	Index		1019

Index 1021

Release v1.4.0. (Installation)

XBee devices allow you to enable wireless connectivity to your projects creating a network of connected devices. They provide features to exchange data with other devices in the network, configure them and control their I/O lines. An application running in an intelligent device can take advantage of these features to monitor and manage the entire network.

Despite the available documentation and configuration tools for working with XBee devices, it is not always easy to develop these kinds of applications.



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

- Support for multiple XBee devices and protocols.
- High abstraction layer provides an easy-to-use workflow.
- Ability to configure local and remote XBee devices of the network.
- Discovery feature finds remote nodes on the same network as the local module.
- Ability to transmit and receive data from any XBee device on the network.
- Ability to manage the General Purpose Input and Output lines of all your XBee devices.
- Ability to send and receive data from other XBee interfaces (Serial, Bluetooth Low Energy and MicroPython).

This portal provides the following documentation to help you with the different development stages of your Python applications using the XBee Python Library.

Getting Started 1

2 Getting Started

## CHAPTER 1

## Requirements

The XBee Python library requires the following components in order to work properly:

- Python 3.6. You can get it from https://www.python.org/getit/
- **PySerial 3**. Install it with pip (pip install pyserial) or refer to the PySerial installation guide for further information about getting PySerial.
- **SRP** Install it with pip (pip install srp).

## CHAPTER 2

### Contents

The XBee Python library documentation is split in different sections:

- Getting Started
- User Documentation
- Examples
- *FAQ*
- Changelog
- API reference

## 2.1 Getting Started

Perform your first steps with the XBee Python library. Learn how to setup your environment and communicate with your XBee devices using the library.

• Get started with XBee Python library

## 2.2 User Documentation

Access detailed information about the different features and capabilities provided by the library and how to use them.

- XBee terminology
- Work with XBee classes
- Configure the XBee device
- Discover the XBee network
- Communicate with XBee devices

- Handle analog and digital IO lines
- Update the XBee
- Log events

## 2.3 Examples

The library includes a good amount of examples that demonstrate most of the functionality that it provides.

• XBee Python samples

### 2.4 FAQ

Find the answer to the most common questions or problems related to the XBee Python library in the FAQ section.

• Frequently Asked Questions (FAQs)

## 2.5 Changelog

• Changelog

## 2.6 API reference

The API reference contains more detailed documentation about the API for developers who are interested in using and extending the library functionality.

• API reference

## 2.6.1 Get started with XBee Python library

This getting started guide describes how to set up your environment and use the XBee Python Library to communicate with your XBee devices. It explains how to configure your modules and write your first XBee Python application.

The guide is split into 3 main sections:

- Install your software
- Configure your XBee modules
- Run your first XBee Python application

#### 2.6.1.1 Install your software

The following software components are required to write and run your first XBee Python application:

- Python 3
- PySerial 3
- SRP

- XBee Python library software
- XCTU

#### Python 3

The XBee Python library requires Python 3. If you don't have Python 3, you can get it from https://www.python.org/getit/.

**Warning:** The XBee Python library is currently only compatible with Python 3.

#### PySerial 3

You must be able to communicate with the radio modules over a serial connection. The XBee Python library uses the **PySerial** module for that functionality.

This module is automatically downloaded when you install the XBee Python library.

#### **SRP**

The XBee Python library uses the **SRP** module to authenticate with XBee devices over Bluetooth Low Energy.

This module is automatically downloaded when you install the XBee Python library.

#### **XBee Python library software**

The best way to install the XBee Python library is with the pip tool (which is what Python uses to install packages). The pip tool comes with recent versions of Python.

To install the library, run this command in your terminal application:

```
$ pip install digi-xbee
```

The library is automatically downloaded and installed in your Python interpreter.

#### Get the source code

The XBee Python library is actively developed on GitHub, where the code is always available. You can clone the repository with:

```
$ git clone git@github.com:digidotcom/xbee-python.git
```

#### **XCTU**

XCTU is a free multi-platform application that enables developers to interact with Digi RF modules through a simple-to-use graphical interface. It includes new tools that make it easy to set up, configure, and test XBee RF modules.

For instructions on downloading and using XCTU, go to:

http://www.digi.com/xctu

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.

#### 2.6.1.2 Configure your XBee modules

You need to configure **two XBee devices**. One module (the sender) sends "Hello XBee World!" using the Python application. The other device (the receiver) receives the message.

To communicate, both devices must be working in the same protocol (802.15.4, Zigbee, DigiMesh, Point-to-Multipoint, or Wi-Fi) and must be configured to operate in the same network.

**Note:** If you are getting started with cellular, you only need to configure one device. Cellular protocol devices are connected directly to the Internet, so there is no network of remote devices to communicate with them. For the cellular protocol, the XBee application demonstrated in the getting started guide differs from other protocols. The cellular protocol sends and reads data from an echo server.

Use XCTU to configure the devices. Plug the devices into the XBee adapters and connect them to your computer's USB or serial ports.

**Note:** For more information about XCTU, see the XCTU User Guide. You can also access the documentation from the Help menu of the tool.

Once XCTU is running, add your devices to the tool and then select them from the **Radio Modules** section. When XCTU is finished reading the device parameters, complete the following steps according to your device type. Repeat these steps to configure your XBee devices using XCTU.

- 802.15.4 devices
- Zigbee devices
- DigiMesh devices
- DigiPoint devices
- Cellular devices
- · Wi-Fi devices

#### 802.15.4 devices

- 1. Click **Load default firmware settings** in the **Radio Configuration** toolbar to load the default values for the device firmware.
- 2. Make sure API mode (API1 or API2) is enabled. To do so, set the **AP** parameter value to **1** (API mode without escapes) or **2** (API mode with escapes).
- 3. Configure **ID** (PAN ID) setting to **CAFE**.
- 4. Configure **CH** (Channel setting) to **C**.
- 5. Click **Write radio settings** in the **Radio Configuration** toolbar to apply the new values to the module.
- 6. Once you have configured both modules, check to make sure they can see each other. Click **Discover radio** modules in the same network, the second button of the device panel in the **Radio Modules** view. The other device must be listed in the **Discovering remote devices** dialog.

**Note:** If the other module is not listed, reboot both devices by pressing the **Reset** button of the carrier board and try adding the device again. If the list is still empty, see the product manual for your device.

#### **Zigbee devices**

- 1. For old Zigbee devices (S2 and S2B), make sure the devices are using **API firmware**. The firmware appears in the **Function** label of the device in the Radio Modules view.
  - One of the devices must be a coordinator Function: Zigbee Coordinator API
  - Digi recommends the other one is a router Function: Zigbee Router AP.

**Note:** If any of the two previous conditions is not satisfied, you must change the firmware of the device. Click the **Update firmware** button of the Radio Configuration toolbar.

- Click Load default firmware settings in the Radio Configuration toolbar to load the default values for the device firmware.
- 3. Do the following:
  - If the device has the **AP** parameter, set it to **1** (API mode without escapes) or **2** (API mode with escapes).
  - If the device has the **CE** parameter, set it to **Enabled** in the coordinator.
- 4. Configure **ID** (PAN ID) setting to **C001BEE**.
- 5. Configure SC (Scan Channels) setting to FFF.
- 6. Click **Write radio settings** in the **Radio Configuration** toolbar to apply the new values to the module.
- 7. Once you have configured both modules, check to make sure they can see each other. Click **Discover radio** modules in the same network, the second button of the device panel in the **Radio Modules** view. The other device must be listed in the **Discovering remote devices** dialog.

**Note:** If the other module is not listed, reboot both devices by pressing the **Reset** button of the carrier board and try adding the device again. If the list is still empty, go to the corresponding product manual for your devices.

#### DigiMesh devices

- Click Load default firmware settings in the Radio Configuration toolbar to load the default values for the device firmware.
- 2. Ensure the API mode (API1 or API2) is enabled. To do so, the **AP** parameter value must be **1** (API mode without escapes) or **2** (API mode with escapes).
- 3. Configure **ID** (PAN ID) setting to **CAFE**.
- 4. Configure **CH** (Operating Channel) to **C**.
- 5. Click Write radio settings in the Radio Configuration toolbar to apply the new values to the module.
- 6. Once you have configured both modules, check to make sure they can see each other. Click **Discover radio** modules in the same network, the second button of the device panel in the **Radio Modules** view. The other device must be listed in the **Discovering remote devices** dialog.

**Note:** If the other module is not listed, reboot both devices by pressing the **Reset** button of the carrier board and try adding the device again. If the list is still empty, go to the corresponding product manual for your devices.

#### **DigiPoint devices**

- 1. Click **Load default firmware settings** in the **Radio Configuration** toolbar to load the default values for the device firmware.
- 2. Ensure the API mode (API1 or API2) is enabled. To do so, the **AP** parameter value must be **1** (API mode without escapes) or **2** (API mode with escapes).
- 3. Configure **ID** (PAN ID) setting to **CAFE**.
- 4. Configure **HP** (Hopping Channel) to **5**.
- 5. Click Write radio settings in the Radio Configuration toolbar to apply the new values to the module.
- 6. Once you have configured both modules, check to make sure they can see each other. Click **Discover radio** modules in the same network, the second button of the device panel in the **Radio Modules** view. The other device must be listed in the **Discovering remote devices** dialog.

**Note:** If the other module is not listed, reboot both devices by pressing the **Reset** button of the carrier board and try adding the device again. If the list is still empty, go to the corresponding product manual for your devices.

#### Cellular devices

- 1. Click **Load default firmware** settings in the Radio Configuration toolbar to load the default values for the device firmware.
- 2. Ensure the API mode (API1 or API2) is enabled. To do so, the **AP** parameter value must be **1** (API mode without escapes) or **2** (API mode with escapes).
- 3. Click **Write radio settings** in the Radio Configuration toolbar to apply the new values to the module.
- 4. Verify the module is correctly registered and connected to the Internet. To do so check that the LED on the development board blinks. If it is solid or has a double-blink, registration has not occurred properly. Registration can take several minutes.

**Note:** In addition to the LED confirmation, you can check the IP address assigned to the module by reading the **MY** parameter and verifying it has a value different than **0.0.0.0**.

#### Wi-Fi devices

- Click Load default firmware settings in the Radio Configuration toolbar to load the default values for the device firmware.
- 2. Ensure the API mode (API1 or API2) is enabled. To do so, the **AP** parameter value must be **1** (API mode without escapes) or **2** (API mode with escapes).
- 3. Connect to an access point:

- 1. Click the Active Scan button.
- 2. Select the desired access point from the list of the Active Scan result dialog.
- 3. If the access point requires a password, type your password.
- 4. Click the **Connect** button and wait for the module to connect to the access point.
- 4. Click **Write radio settings** in the Radio Configuration toolbar to apply the new values to the module.
- 5. Verify the module is correctly connected to the access point by checking the IP address assigned to the module by reading the **MY** parameter and verifying it has a value different than **0.0.0.0**.

#### 2.6.1.3 Run your first XBee Python application

The XBee Python application demonstrated in the guide broadcasts the message *Hello XBee World!* from one of the devices connected to your computer (the sender) to all remote devices on the same network as the sender. Once the message is sent, the receiver XBee module must receive it. You can use XCTU to verify receipt.

The commands to be executed depend on the protocol of the XBee devices. Follow the corresponding steps depending on the protocol of your XBee devices.

- Zigbee, DigiMesh, DigiPoint or 802.15.4 devices
- · Wi-Fi devices
- Cellular devices

#### Zigbee, DigiMesh, DigiPoint or 802.15.4 devices

Follow these steps to send the broadcast message and verify that it is received successfully:

- 1. First, prepare the *receiver* XBee device in XCTU to verify that the broadcast message sent by the *sender* device is received successfully. Follow these steps to do so:
  - 1. Launch XCTU.
  - 2. Add the *receiver* module to XCTU.
  - 3. Click **Open the serial connection with the radio module** to switch to **Consoles working mode** and open the serial connection. This allows you to see the data when it is received.
- 2. Open the Python interpreter and write the application commands.
  - 1. Import the XBeeDevice class by executing the following command:

```
> from digi.xbee.devices import XBeeDevice
```

2. Instantiate a generic XBee device:

```
> device = XBeeDevice("COM1", 9600)
```

**Note:** Remember to replace the COM port with the one your *sender* XBee device is connected to. In UNIX-based systems, the port usually starts with /dev/tty.

3. Open the connection with the device:

```
> device.open()
```

4. Send the Hello XBee World! broadcast message.

```
> device.send_data_broadcast("Hello XBee World!")
```

5. Close the connection with the device:

```
> device.close()
```

3. Verify that the message is received by the *receiver* XBee in XCTU. An **RX** (**Receive**) **frame** should be displayed in the **Console log** with the following information:

Start delimiter	7E
Length	Depends on the XBee protocol
Frame type	Depends on the XBee protocol
16/64-bit source address	XBee sender's 16/64-bit address
Options	02
RF data/Received data	48 65 6C 6C 6F 20 58 42 65 65 20 57 6F 72 6C 64 21

#### Wi-Fi devices

Wi-Fi devices send broadcast data using the <code>send\_ip\_data\_broadcast()</code> command instead of the <code>send\_data\_broadcast()</code> one. For that reason, you must instantiate a <code>WiFiDevice</code> instead of a generic <code>XBeeDevice</code> to execute the proper command.

Follow these steps to send the broadcast message and verify that it is received successfully:

- 1. First, prepare the *receiver* XBee device in XCTU to verify that the broadcast message sent by the *sender* device is received successfully by the *receiver* device.
  - 1. Launch XCTU.
  - 2. Add the receiver module to XCTU.
  - 3. Click **Open the serial connection with the radio module** to switch to **Consoles working mode** and open the serial connection. This allows you to see the data when it is received.
- 2. Open the Python interpreter and write the application commands.
  - 1. Import the WiFiDevice class by executing the following command:

```
> from digi.xbee.devices import WiFiDevice
```

2. Instantiate a Wi-Fi XBee device:

```
> device = WiFiDevice("COM1", 9600)
```

**Note:** Remember to replace the COM port with the one your *sender* XBee device is connected to. In UNIX-based systems, the port usually starts with /dev/tty.

3. Open the connection with the device:

```
> device.open()
```

4. Send the Hello XBee World! broadcast message.

```
> device.send_ip_data_broadcast(9750, "Hello XBee World!")
```

5. Close the connection with the device:

```
> device.close()
```

3. Verify that the message is received by the *receiver* XBee in XCTU. An **RX IPv4 frame** should be displayed in the **Console log** with the following information:

Start delimiter	7E
Length	00 1C
Frame type	B0
IPv4 source address	XBee Wi-Fi sender's IP address
16-bit dest port	26 16
16-bit source port	26 16
Protocol	00
Status	00
RF data	48 65 6C 6C 6F 20 58 42 65 65 20 57 6F 72 6C 64 21

#### Cellular devices

Cellular devices are connected directly to the Internet, so there is no network of remote devices to communicate with them. For cellular protocol, the application demonstrated in this guide differs from other protocols.

The application sends and reads data from an echo server. Follow these steps to execute it:

- 1. Open the Python interpreter and write the application commands.
  - 1. Import the CellularDevice, IPProtocol and IPv4Address classes:

```
> from digi.xbee.devices import CellularDevice
> from digi.xbee.models.protocol import IPProtocol
> from ipaddress import IPv4Address
```

2. Instantiate a cellular XBee device:

```
> device = CellularDevice("COM1", 9600)
```

**Note:** Remember to replace the COM port by the one your Cellular XBee device is connected to. In UNIX-based systems, the port usually starts with /dev/tty.

3. Open the connection with the device:

```
> device.open()
```

4. Send the *Hello XBee World!* message to the echo server with IP 52.43.121.77 and port 11001 using the *TCP IP* protocol.

```
> device.send_ip_data(IPv4Address("52.43.121.77"), 11001, IPProtocol.TCP,

→"Hello XBee World!")
```

5. Read and print the response from the echo server. If response cannot be received, print ERROR.

```
> ip_message = device.read_ip_data()
> print(ip_message.data.decode("utf8") if ip_message is not None else "ERROR")
```

6. Close the connection with the device:

```
> device.close()
```

### 2.6.2 XBee terminology

This section covers basic XBee concepts and terminology. The XBee Python library manual refers to these concepts frequently, so it is important to understand these concepts.

#### 2.6.2.1 RF modules

A radio frequency (RF) module is a small electronic circuit used to transmit and receive radio signals on different frequencies. Digi produces a wide variety of RF modules to meet the requirements of almost any wireless solution, such as long-range, low-cost, and low power modules.

#### 2.6.2.2 XBee RF modules

XBee is the brand name of a family of RF modules produced by Digi International Inc. XBee RF modules are modular products that make it easy and cost-effective to deploy wireless technology. Multiple protocols and RF features are available, giving customers enormous flexibility to choose the best technology for their needs.

The XBee RF modules are available in three form factors: Through-Hole, Surface Mount, and Micro, with different antenna options. Almost all modules are available in the Through-Hole form factor and share the same footprint.



#### 2.6.2.3 Radio firmware

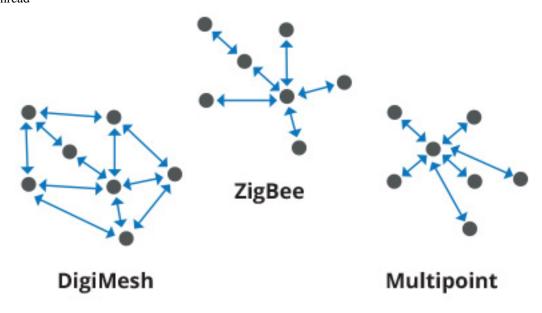
Radio firmware is the program code stored in the radio module's persistent memory that provides the control program for the device. From XCTU or the local web interface of the XBee Gateway, you can update or change the firmware of the local XBee module or any other module connected to the same network. This is a common task when changing the role of the device or updating to the latest version of the firmware.

#### 2.6.2.4 Radio communication protocols

A radio communication protocol is a set of rules for data exchange between radio devices. An XBee module supports a specific radio communication protocol depending on the module and its radio firmware.

Following is the complete list of protocols supported by the XBee radio modules:

- IEEE 802.15.4
- Zigbee
- Zigbee Smart Energy
- DigiMesh (Digi proprietary)
- ZNet
- IEEE 802.11 (Wi-Fi)
- Point-to-multipoint (Digi proprietary)
- XSC (XStream compatibility)
- Cellular
- · Thread



**Note:** Not all XBee devices can run all these communication protocols. The combination of XBee hardware and radio firmware determines the protocol that an XBee device can execute. Refer to the XBee RF Family Comparison Matrix for more information about the available XBee RF modules and the protocols they support.

#### 2.6.2.5 Radio module operating modes

The operating mode of an XBee radio module establishes the way a user, or any microcontroller attached to the XBee, communicates with the module through the Universal Asynchronous Receiver/Transmitter (UART) or serial interface.

Depending on the firmware and its configuration, the radio modules can work in three different operating modes:

- Application Transparent (AT) operating mode
- API operating mode
- API escaped operating mode

In some cases, the operating mode of a radio module is established by the firmware version and the firmware's AP setting. The module's firmware version determines whether the operating mode is AT or API. The firmware's AP setting determines if the API mode is escaped ( $\mathbf{AP} = 2$ ) or not ( $\mathbf{AP} = 1$ ). In other cases, the operating mode is only

determined by the AP setting, which allows you to configure the mode to be AT ( $\mathbf{AP} = 0$ ), API ( $\mathbf{AP} = 1$ ) or API escaped ( $\mathbf{AP} = 2$ ).

#### Application Transparent (AT) operating mode

In Application Transparent (AT) or transparent operating mode, all serial data received by the radio module is queued up for RF transmission. When the module receives RF data, it sends the data out through the serial interface.

To configure an XBee module operating in AT, put the device in command mode to send the configuration commands.

#### Command mode

When the radio module is working in AT operating mode, configure settings using the command mode interface.

To enter command mode, send the 3-character command sequence through the serial interface of the radio module, usually +++, within one second. Once the command mode has been established, the module sends the reply OK, the command mode timer starts, and the radio module can receive AT commands.

The structure of an AT command follows this format:

```
AT[ASCII command][Space (optional)][Parameter (optional)][Carriage return]
```

#### Example:

ATNI MyDevice\r

If no valid AT commands are received within the command mode timeout, the radio module automatically exits command mode. You can also exit command mode issuing the CN command (Exit Command mode).

#### **API** operating mode

Application Programming Interface (API) operating mode is an alternative to AT operating mode. API operating mode requires that communication with the module through a structured interface; that is, data communicated in API frames.

The API specifies how commands, command responses, the module sends and receives status messages using the serial interface. API operation mode enables many operations, such as the following:

- Configure the XBee device itself.
- Configure remote devices in the network.
- Manage data transmission to multiple destinations.
- Receive success/failure status of each transmitted RF packet.
- Identify the source address of each received packet.

Depending on the AP parameter value, the device can operate in one of two modes: API ( $\mathbf{AP} = 1$ ) or API escaped ( $\mathbf{AP} = 2$ ) operating mode.

#### API escaped operating mode

API escaped operating mode ( $\mathbf{AP} = 2$ ) works similarly to API mode. The only difference is that when working in API escaped mode, some bytes of the API frame specific data must be escaped.

Use API escaped operating mode to add reliability to the RF transmission, which prevents conflicts with special characters such as the start-of-frame byte (0x7E). Since 0x7E can only appear at the start of an API packet, if 0x7E is

received at any time, you can assume that a new packet has started regardless of length. In API escaped mode, those special bytes are escaped.

#### **Escape characters**

When sending or receiving an API frame in API escaped mode, you must escape (flag) specific data values so they do not interfere with the data frame sequence. To escape a data byte, insert 0x7D and follow it with the byte being escaped, XOR'd with 0x20.

The following data bytes must be escaped:

• 0x7E: Frame delimiter

0x7D: Escape0x11: XON0x13: XOFF

#### 2.6.2.6 API frames

An API frame is the structured data sent and received through the serial interface of the radio module when it is configured in API or API escaped operating modes. API frames are used to communicate with the module or with other modules in the network.

An API frame has the following structure:

Start Delimiter (Byte 1)		ngth es 2-3)	Frame Data (Bytes 4- <i>n</i> )	Checksum (Byte n + 1)
0x7E	MSB	LSB	API-specific Structure	1 Byte

Start	This field is always 0x7E.
delim-	
iter	
Length	The length field has a two-byte value that specifies the number of bytes that are contained in the frame
	data field. It does not include the checksum field.
Frame	The content of this field is composed by the API identifier and the API identifier specific data. Depend-
Data	ing on the API identifier (also called API frame type), the content of the specific data changes.
Check-	Byte containing the hash sum of the API frame bytes.
sum	

In API escaped mode, some bytes in the Length, Frame Data and Checksum fields must be escaped.



#### 2.6.2.7 AT settings or commands

The firmware running in the XBee RF modules contains a group of settings and commands that you can configure to change the behavior of the module or to perform any related action. Depending on the protocol, the number of settings and meanings vary, but all the XBee RF modules can be configured with AT commands.

All the firmware settings or commands are identified with two ASCII characters and some applications and documents refer to them as **AT settings** or **AT commands**.

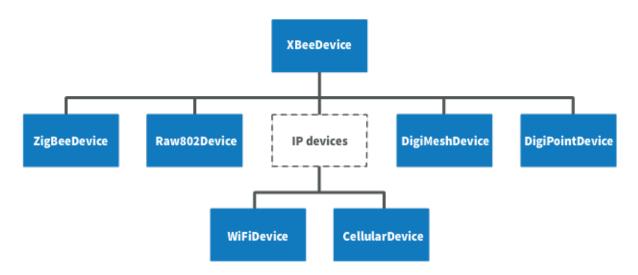
The configuration process of the AT settings varies depending on the operating mode of the XBee RF module.

- AT operating mode. In this mode, you must put the module in a special mode called command mode, so it can receive AT commands. For more information about configuring XBee RF modules working in AT operating mode, see *Application Transparent (AT) operating mode*.
- **API operating mode**. To configure or execute AT commands when the XBee RF module operates in API mode, you must generate an AT command API frame containing the AT setting identifier and the value of that setting, and send it to the XBee RF module. For more information about API frames, see *API frames*.

#### 2.6.3 Work with XBee classes

When working with the XBee Python Library, start with an XBee device object that represents a physical module. A physical XBee device is the combination of hardware and firmware. Depending on that combination, the device runs a specific wireless communication protocol such as Zigbee, 802.15.4, DigiMesh, Wi-Fi, or cellular. An XBeeDevice class represents the XBee module in the API.

Most of the protocols share the same features and settings, but there are some differences between them. For that reason, the XBee Python Library also includes a set of classes that represent XBee devices running different communication protocols. The XBee Python Library supports one XBee device class per protocol, as follows:



- XBee Zigbee device (ZigBeeDevice)
- XBee 802.15.4 device (Raw802Device)
- XBee DigiMesh device (DigiMeshDevice)
- XBee Point-to-multipoint device (DigiPointDevice)
- XBee IP devices (This is a non-instantiable class)

- XBee Cellular device (CellularDevice)
- XBee Wi-Fi device (WiFiDevice)

All these XBee device classes allow you to configure the physical XBee device, communicate with the device, send data to other nodes on the network, receive data from remote devices, and so on. Depending on the class, you may have additional methods to execute protocol-specific features or similar methods.

To work with the API and perform actions involving the physical device, you must instantiate a generic XBeeDevice object or one that is protocol-specific. This documentation refers to the XBeeDevice object generically when describing the different features, but they can be applicable to any XBee device class.

#### 2.6.3.1 Instantiate an XBee device

When you are working with the XBee Python Library, the first step is to instantiate an XBee device object. The API works well using the generic XBeeDevice class, but you can also instantiate a protocol-specific XBee device object if you know the protocol your physical XBee device is running.

An XBee device is represented as either **local** or **remote** in the XBee Python Library, depending upon how you communicate with the device.

#### Local XBee device

A local XBee device is the object in the library representing the device that is physically attached to your PC through a serial or USB port. The classes you can instantiate to represent a local device are listed in the following table:

Class	Description
XBeeDevice	Generic object, protocol-independent
ZigBeeDevice	Zigbee protocol
Raw802Device	802.15.4 protocol
DigiMeshDevice	DigiMesh protocol
DigiPointDevice	Point-to-multipoint protocol
CellularDevice	Cellular protocol
WiFiDevice	Wi-Fi protocol

To instantiate a generic or protocol-specific XBee device, you need to provide the following two parameters:

- Serial port name
- Serial port baud rate

#### Instantiate a local XBee device

```
[...]
xbee = XBeeDevice("COM1", 9600)
[...]
```

#### Remote XBee device

Remote XBee device objects represent remote nodes of the network. These are XBee devices that are not attached to your PC but operate in the same network as the attached (local) device.

**Warning:** When working with remote XBee devices, it is very important to understand that you cannot communicate directly with them. You need to provide a local XBee device operating in the same network that acts as bridge between your serial port and the remote node.

Managing remote devices is similar to managing local devices, but with limitations. You can configure them, handle their IO lines, and so on, in the same way you manage local devices. Local XBee devices have several methods for sending data to remote devices, but the remote devices cannot use these methods because they are already remote. Therefore, a remote device cannot send data to another remote device.

In the local XBee device instantiation, you can choose between instantiating a generic remote XBee device object or a protocol-specific remote XBee device. The following table lists the remote XBee device classes:

Class	Description
RemoteXBeeDevice	Generic object, protocol independent
RemoteZigBeeDevice	Zigbee protocol
RemoteRaw802Device	802.15.4 protocol
RemoteDigiMeshDevice	DigiMesh protocol
RemoteDigiPointDevice	Point-to-multipoint protocol

**Note:** XBee Cellular and Wi-Fi protocols do not support remote devices.

To instantiate a remote XBee device object, you need to provide the following parameters:

- Local XBee device attached to your PC that serves as the communication interface.
- 64-bit address of the remote device.

RemoteRaw802Device objects can be also instantiated by providing the local XBee device attached to your PC and the **16-bit address** of the remote device.

#### Instantiate a remote XBee device

20

The local device must also be the same protocol for protocol-specific remote XBee devices.

#### 2.6.3.2 Open the XBee device connection

Before trying to communicate with the local XBee device attached to your PC, you need to open its communication interface, which is typically a serial/USB port. Use the open () method of the instantiated XBee device, and you can then communicate and configure the device.

Remote XBee devices do not have an open method. They use a local XBee device as the connection interface. If you want to perform any operation with a remote XBee device you must open the connection of the associated local device.

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
# Open the device connection.
local_xbee.open()
[...]
```

The open () method may fail for the following reasons:

- All the possible errors are caught as XBeeException:
  - If there is any problem with the communication, throwing a TimeoutException.
  - If the operating mode of the device is not API or API\_ESCAPE, throwing an InvalidOperatingModeException.
  - There is an error writing to the XBee interface, or device is closed, throwing a generic XBeeException.

The open () action performs some other operations apart from opening the connection interface of the device. It reads the device information (reads some sensitive data from it) and determines the operating mode of the device.

Use force\_settings=True as open() method parameter, to reconfigure the XBee serial settings (baud rate, data bits, stop bits, etc.) to those specified in the XBee object constructor.

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)

# Open the connection using constructor parameters: 9600 8N1.
# This reconfigures the XBee if its serial settings do not match.
local_xbee.open(force_settings=True)
[...]
```

#### Example: Recover XBee serial communication

The XBee Python Library includes a sample application that displays how to recover the serial connection with a local XBee. It can be located in the following path:

examples/configuration/Recover Serial Connection/Recover Serial Connection.py

#### Read device information

The read device information process reads the following parameters from the local or remote XBee device and stores them inside. You can then access parameters at any time, calling their corresponding getters.

- 64-bit address
- 16-bit address
- · Node identifier
- · Firmware version
- · Hardware version
- IPv4 address (only for cellular and Wi-Fi modules)

• IMEI (only for cellular modules)

The read process is performed automatically in local XBee devices when opening them with the open() method. If remote XBee devices cannot be opened, you must use read\_device\_info() to read their device information.

#### Initialize a remote XBee device

The read\_device\_info() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - If the operating mode of the device is not API or API\_ESCAPE, throwing an InvalidOperatingModeException.
  - If the response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, or device is closed, throwing a generic XBeeException.

**Note:** Although the readDeviceInfo method is executed automatically in local XBee devices when they are open, you can issue it at any time to refresh the information of the device.

#### Get device information

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Get the 64-bit address of the device.
addr_64 = device.get_64bit_addr()

# Get the node identifier of the device.
node_id = device.get_node_id()

# Get the hardware version of the device.
hardware_version = device.get_hardware_version()

# Get the firmware version of the device.
firmware_version = device.get_firmware_version()
```

The read device information process also determines the communication protocol of the local or remote XBee device object. This is typically something you need to know beforehand if you are not using the generic XBeeDevice object.

However, the API performs this operation to ensure that the class you instantiated is the correct one. So, if you instantiated a Zigbee device and the open () process realizes that the physical device is actually a DigiMesh device, you receive an XBeeDeviceException indicating the device mismatch.

You can retrieve the protocol of the XBee device from the object executing the corresponding getter.

#### Get the XBee protocol

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Get the protocol of the device.
protocol = local_xbee.get_protocol()
```

#### **Device operating mode**

The open () process also reads the operating mode of the physical local device and stores it in the object. As with previous settings, you can retrieve the operating mode from the object at any time by calling the corresponding getter.

#### Get the operating mode

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Get the operating mode of the device.
operating_mode = local_xbee.get_operating_mode()
```

Remote devices do not have an open () method, so you receive UNKNOWN when retrieving the operating mode of a remote XBee device.

The XBee Python Library supports two operating modes for local devices:

- API
- API with escaped characters

This means that AT (transparent) mode is not supported by the API. So, if you try to execute the open() method in a local device working in AT mode, you get an XBeeException caused by an InvalidOperatingModeException.

#### 2.6.3.3 Close the XBee device connection

You must call the close () method each time you finish your XBee application. You can use this in the finally block or something similar.

If you don't do this, you may have problems with the packet listener being executed in a separate thread.

This method guarantees that the listener thread will be stopped and the serial port will be closed.

#### Close the connection

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)

try:
    xbee.open()
    [...]

finally:
    if xbee is not None and xbee.is_open():
        xbee.close()
```

**Note:** Remote XBee devices cannot be opened, so they cannot be closed either. To close the connection of a remote device you need to close the connection of the local associated device.

### 2.6.4 Configure the XBee device

One of the main features of the XBee Python Library is the ability to configure the parameters of local and remote XBee devices and execute some actions or commands on them.

To apply a complete configuration profile see *Apply an XBee profile*.

**Warning:** The values set on the different parameters are not persistent through subsequent resets unless you store those changes in the device. For more information, see *Write configuration changes*.

#### 2.6.4.1 Read and set common parameters

Local and remote XBee device objects provide a set of methods to get and set common parameters of the device. Some of these parameters are saved inside the XBee device object, and a cached value is returned when the parameter is requested. Other parameters are read directly from the physical XBee device when requested.

#### **Cached parameters**

Some parameters in an XBee device are used or requested frequently. To avoid the overhead of those parameters being read from the physical XBee device every time they are requested, they are saved inside the XBeeDevice object being returned when the getters are called.

The following table lists cached parameters and their corresponding getters:

Parameter	Method
64-bit address	get_64bit_addr()
16-bit address	get_16bit_addr()
Node identifier	get_node_id()
Firmware version	get_firmware_version()
Hardware version	get_hardware_version()
Role	get_role()

Local XBee devices read and save previous parameters automatically when opening the connection of the device. In remote XBee devices, you must issue the read\_device\_info() method to initialize the parameters.

You can refresh the value of those parameters (that is, read their values and update them inside the XBee device object) at any time by calling the read\_device\_info() method.

Method	Description
read_device_info(init=	False tes cache parameters reading them from the XBee: If init is True it reads all
	values, else only those not initialized.

#### Refresh cached parameters

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
# Refresh the cached values.
local_xbee.refresh_device_info()
[...]
```

The read\_device\_info() method may fail for the following reasons:

- There is a timeout getting any of the device parameters, throwing a TimeoutException.
- The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
- The response of the command is not valid, throwing an ATCommandException.
- There is an error writing to the XBee interface, or device is closed, throwing a generic XBeeException.

All the cached parameters but the Node Identifier do not change; therefore, they cannot be set. For the Node Identifier, there is a method within all the XBee device classes that allows you to change it:

Method	Description
set_node_id	(String) ies the new Node Identifier of the device. This method configures the physical XBee device
	with the provided Node Identifier and updates the cached value with the one provided.

#### Non-cached parameters

The following non-cached parameters have their own methods to be configured within the XBee device classes:

• **Destination Address**: This setting specifies the default 64-bit destination address of a module that is used to report data generated by the XBee device (that is, IO sampling data). This setting can be read and set.

Method	Description	
<b>get_dest_address</b> () Returns the 64-bit address of the device that data will be report		
set_dest_address(XBee64BitAddress) Specifies the 64-bit address of the device where the data will be re-		
	ported.	

• PAN ID: This is the ID of the Personal Area Network the XBee device is operating in. This setting can be read and set.

Method	Description	
get_pan_id()	Returns a byte array containing the ID of the Personal Area Network where the XBee	
	device is operating.	
set_pan_id(Bytearray)pecifies the value in byte array format of the PAN ID where the XBee device should		
work.		

• Power level: This setting specifies the output power level of the XBee device. This setting can be read and set.

Method	Description	
get_power_level()	Returns a <b>PowerLevel</b> enumeration entry indicating the power level of the XBee	
	device.	
set_power_level(PowerLevel) ecifies a PowerLevel enumeration entry containing the desired output level		
	of the XBee device.	

#### Configure non-cached parameters

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Set the destination address of the device.
dest_address = XBee64BitAddress.from_hex_string("0013A20040XXXXXX")
local_xbee.set_dest_address(dest_address)

# Read the operating PAN ID of the device.
dest_addr = local_xbee.get_dst_address()

# Read the operating PAN ID of the device.
pan_id = local_xbee.get_pan_id()

# Read the output power level.
p_level = local_xbee.get_power_level()

[...]
```

All the previous getters and setters of the different options may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

#### Example: Common parameters

The XBee Python Library includes a sample application that displays how to get and set common parameters. It can be located in the following path:

examples/configuration/ManageCommonParametersSample

#### 2.6.4.2 Read, set and execute other parameters

If you want to read or set a parameter that does not have a custom getter or setter within the XBee device object, you can do so. All the XBee device classes (local or remote) include two methods to get and set any AT parameter, and a third one to run a command in the XBee device.

#### Get a parameter

You can read the value of any parameter of an XBee device using the <code>get\_parameter()</code> method provided by all the XBee device classes. Use this method to get the value of a parameter that does not have its getter method within the XBee device object.

Method	Description	
get_parameter(String) ecifies the AT command (string format) to retrieve its value. The method returns the value		
of the parameter in a byte array.		

#### Get a parameter from the XBee device

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Get the value of the Sleep Time (SP) parameter.
sp = local_xbee.get_parameter("SP")
[...]
```

The get\_parameter() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

#### Example: Set and get parameters

The XBee Python Library includes a sample application that displays how to get and set parameters using the methods explained previously. It can be located in the following path:

examples/configuration/SetAndGetParametersSample

#### Set a parameter

To set a parameter that does not have its own setter method, you can use the set\_parameter() method provided by all the XBee device classes.

Method	Description	
set_parameter(String,	Specifies the AT command (String format) to be set in the device and a byte array	
Bytearray)	containing the value of the parameter.	

#### Set a parameter in the XBee device

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Configure the Node ID using the set_parameter() method.
local_xbee.set_parameter("NI", bytearray("Yoda", 'utf8'))
[...]
```

The set\_parameter() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

#### Example: Set and get parameters

The XBee Python Library includes a sample application that displays how to get and set parameters using the methods explained previously. It can be located in the following path:

examples/configuration/SetAndGetParametersSample

#### **Execute a command**

There are other AT parameters that cannot be read or written. They are actions that are executed by the XBee device. The XBee Python library has several commands that handle most common executable parameters, but to run a parameter that does not have a custom command, you can use the <code>execute\_command()</code> method provided by all the XBee device classes.

Method	Description	
execute_command(String)	Specifies the AT command (String format) to be run in the device.	

#### Run a command in the XBee device

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
# Run the apply changes command.
```

(continues on next page)

(continued from previous page)

```
local_xbee.execute_command("AC")
[...]
```

The execute\_command() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

#### 2.6.4.3 Apply configuration changes

By default, when you perform any configuration on a local or remote XBee device, the changes are automatically applied. However, there could be some scenarios when you want to configure different settings or parameters of a device and apply the changes at the end when everything is configured. For that purpose, the XBeeDevice and RemoteXBeeDevice objects provide some methods that allow you to manage when to apply configuration changes.

Method	Description	Notes
en-	Specifies whether the changes on settings and	The apply configuration changes flag is enabled
able_apply_changes(Bootkern) are applied when set.		by default.
is_apply_changes_kenabled()) hether the XBee device is config-		
ured to apply parameter changes when they		
	are set.	
ap-	Applies the changes on parameters that were	This method is useful when the XBee device is
ply_changes()	already set but are pending to be applied.	configured to not apply changes when they are
		set.

#### Apply configuration changes

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
# Check if device is configured to apply changes.
apply_changes_enabled = local_xbee.is_apply_changes_enabled()
# Configure the device not to apply parameter changes automatically.
if apply_changes_enabled:
    local_xbee.enable_apply_changes(False)

# Set the PAN ID of the XBee device to BABE.
local_xbee.set_pan_id(utils.hex_string_to_bytes("BABE"))
# Perform other configurations.
[...]
```

(continues on next page)

(continued from previous page)

```
# Apply changes.
local_xbee.apply_changes()
[...]
```

The apply\_changes () method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

#### 2.6.4.4 Write configuration changes

If you want configuration changes performed in an XBee device to persist through subsequent resets, you need to write those changes in the device. Writing changes means that the parameter values configured in the device are written to the non-volatile memory of the XBee device. The module loads the parameter values from non-volatile memory every time it is started.

The XBee device classes (local and remote) provide a method to write (save) the parameter modifications in the XBee device memory so they persist through subsequent resets: write\_changes().

#### Write configuration changes

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Set the PAN ID of the XBee device to BABE.
local_xbee.set_pan_id(utils.hex_string_to_bytes("BABE"))

# Perform other configurations.
[...]

# Apply changes.
local_xbee.apply_changes()

# Write changes.
local_xbee.write_changes()
[...]
```

The write\_changes () method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.

- There is an error writing to the XBee interface, throwing a generic XBeeException.

#### 2.6.4.5 Reset the device

It may be necessary to reset the XBee device when the system is not operating properly or you are initializing the system. All the XBee device classes of the XBee API provide the reset () method to perform a software reset on the local or remote XBee module.

In local modules, the reset () method blocks until a confirmation from the module is received, which usually takes one or two seconds. Remote modules do not send any kind of confirmation, so the method does not block when resetting them.

#### Reset the module

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
# Reset the module.
local_xbee.reset()
[...]
```

The reset () method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

## Example: Reset module

The XBee Python Library includes a sample application that shows you how to perform a reset on your XBee device. The example is located in the following path:

examples/configuration/ResetModuleSample

# 2.6.4.6 Configure Wi-Fi settings

Unlike other protocols such as Zigbee or DigiMesh where devices are connected to each other, the XBee Wi-Fi protocol requires that the module is connected to an access point in order to communicate with other TCP/IP devices.

This configuration and connection with access points can be done using applications such as XCTU; however, the XBee Python Library includes a set of methods to configure the network settings, scan access points, and connect to an access point.

# Example: Configure Wi-Fi settings and connect to an access point

The XBee Python Library includes a sample application that demonstrates how to configure the network settings of a Wi-Fi device and connect to an access point. You can locate the example in the following path: **examples/configuration/ConnectToAccessPointSample** 

# Configure IP addressing mode

Before connecting your Wi-Fi module to an access point, you must decide how to configure the network settings using the IP addressing mode option. The supported IP addressing modes are contained in an enumerator called IPAddressingMode. It allows you to choose between:

- DHCP
- STATIC

Method	Description
set_ip_addressing_mode(IPAddressingMode)	Sets the IP addressing mode of the Wi-Fi module. De-
	pending on the provided mode, network settings are
	configured differently:
	• <b>DHCP</b> : Network settings are assigned by a server.
	• STATIC: Network settings must be provided
	manually one by one.

# Configure IP addressing mode

```
[...]
# Instantiate an XBee device object.
local_xbee = WiFiDevice("COM1", 9600)
local_xbee.open()
# Configure the IP addressing mode to DHCP.
local_xbee.set_ip_addressing_mode(IPAddressingMode.DHCP)
# Save the IP addressing mode.
local_xbee.write_changes()
[...]
```

The set\_ip\_addressing\_mode() method may fail for the following reasons:

- There is a timeout setting the IP addressing parameter, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# Configure IP network settings

32

Like any TCP/IP protocol device, the XBee Wi-Fi modules have the IP address, subnet mask, default gateway and DNS settings that you can get at any time using the XBee Python Library.

Unlike some general configuration settings, these parameters are not saved inside the WiFiDevice object. Every time you request the parameters, they are read directly from the Wi-Fi module connected to the computer. The following parameters are used in the configuration of the TCP/IP protocol:

Parameter	Method
IP address	get_ip_address()
Subnet mask	get_mask_address()
Gateway IP	get_gateway_address()
DNS address	get_dns_address()

# Read IP network settings

```
# Instantiate an XBee device object.
local_xbee = WiFiDevice("COM1", 9600)
local_xbee.open()

# Configure the IP addressing mode to DHCP.
local_xbee.set_ip_addressing_mode(IPAddressingMode.DHCP)

# Connect to access point with SSID 'My SSID' and password 'myPassword'
local_xbee.connect_by_ssid("My SSID", "myPassword")

# Display the IP network settings that were assigned by the DHCP server.
print("- IP address: %s" % local_xbee.get_ip_address())
print("- Subnet mask: %s" % local_xbee.get_mask_address())
print("- Gateway IP address: %s" % local_xbee.get_gateway_address())
print("- DNS IP address: %s" % local_xbee.get_dns_address())
[...]
```

You can also change those settings when the module has static IP configuration with the following methods:

Parameter	Method
IP address	set_ip_addr()
Subnet mask	set_mask_address()
Gateway IP	set_gateway_address()
DNS address	set_dns_address()

## 2.6.4.7 Configure Bluetooth settings

Newer XBee3 devices have a Bluetooth® Low Energy (BLE) interface that enables you to connect your XBee device to another device such as a cellphone. The XBee device classes (local and remote) offer some methods that allow you to:

- Enable and disable Bluetooth
- Configure the Bluetooth password
- Read the Bluetooth MAC address

## **Enable and disable Bluetooth**

Before connecting to your XBee device over Bluetooth Low Energy, you first have to enable this interface. The XBee Python Library provides a couple of methods to enable or disable this interface:

Method	Description
enable_bluetooth()	Enables the Bluetooth Low Energy interface of your XBee device.
disable_bluetooth()	Disables the Bluetooth Low Energy interface of your XBee device.

## Enabling and disabling the Bluetooth interface

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Enable the Bluetooth interface.
local_xbee.enable_bluetooth()

[...]

# Disable the Bluetooth interface.
local_xbee.disable_bluetooth()
```

These methods may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

## Configure the Bluetooth password

Once you have enabled the Bluetooth Low Energy, you must configure the password you will use to connect to the device over that interface (if not previously done). For this purpose, the API offers the following method:

Method	Description
update bluetooth password(String)	Specifies the new Bluetooth password of the XBee device.

# Configuring or changing the Bluetooth password

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

new_password = "myBluetoothPassword" # Do not hard-code it in the app!

# Configure the Bluetooth password.
local_xbee.update_bluetooth_password(new_password)
```

(continues on next page)

[...]

The update\_bluetooth\_password method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

**Warning:** Never hard-code the Bluetooth password in the code, a malicious person could decompile the application and find it out.

#### Read the Bluetooth MAC address

Another method that the XBee Java Library provides is get\_bluetooth\_mac\_addr(), which returns the EUI-48 Bluetooth MAC address of your XBee device in a format such as "00112233AABB".

## Reading the Bluetooth MAC address

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
print("The Bluetooth MAC address is: %s" % local_xbee.get_bluetooth_mac_addr())
[...]
```

The get bluetooth mac addr method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# 2.6.5 Discover the XBee network

Several XBee modules working together and communicating with each other form a network. XBee networks have different topologies and behaviors depending on the protocol of the XBee nodes that form it.

The XBee Python Library includes a class, called XBeeNetwork, that represents the set of nodes forming the actual XBee network. This class allows you to perform some operations related to the nodes.

**Note:** There are XBeeNetwork subclasses for different protocols which correspond to the XBeeDevice subclasses:

- XBee Zigbee network (ZigBeeNetwork)
- XBee 802.15.4 network (Raw802Network)
- XBee DigiMesh network (DigiMeshNetwork)
- XBee DigiPoint network (DigiPointNetwork)

**Warning:** Because XBee Cellular and Wi-Fi module protocols are directly connected to the Internet and do not share a connection, these protocols do not support XBee networks.

The XBee network object can be retrieved from a local XBee after it has been opened with the method get\_network().

#### Retrieve the XBee network

```
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()

# Get the network.
xnet = xbee.get_network()
[...]
```

A main feature of the XBeeNetwork class is the ability to discover the XBee nodes that form the network and store them in a internal list. The XBeeNetwork object provides the following operations related to the XBee discovery feature:

- · Discovery types
- Deep discovery
- Standard discovery
- Discover the network
- Access discovered nodes
- Access connections between nodes
- Add and remove nodes manually
- Listen to network modification events

# 2.6.5.1 Discovery types

There are two different types of discovery processes available in this API:

- *Deep discovery* finds network nodes and connections between them (including quality) even if they are sleeping. It also allows to establish a number of rounds to continually explore the network.
- Standard discovery only identifies network nodes. It may not discover sleeping nodes.

See *Discover the network* to know how to launch a deep or standard discovery process.

**Note:** In 802.15.4, both (deep and standard discovery) are the same and none discover the node connections nor their quality. The difference is the possibility of running more than one round using a deep discovery.

## 2.6.5.2 Deep discovery

This discovery process finds network nodes and their connections including the quality. It asks each node for its neighbors and retrieves information about the signal quality between them.

This mechanism also discovers sleeping nodes.

It is possible to configure the discovery process to run a specific number of times or even endlessly. Each discovery round is called a scan.

# Deep discovery modes

This mode establishes the way the network deep discovery process is performed. Available modes are defined in the NeighborDiscoveryMode enumeration:

- Cascade (NeighborDiscoveryMode.CASCADE): The discovery of the neighbors of a node is requested once the previous request finishes. This means that just one discovery process is running at the same time. This mode is recommended for large networks, it might be a slower method but it generates less traffic than 'Flood'.
- Flood (NeighborDiscoveryMode.FLOOD): The discovery of the neighbors of a node is requested when the node is found in the network. This means that several discovery processes might be running at the same time. This might be a faster method, but it generates a lot of traffic and might saturate the network.

The default discovery mode is **Cascade**. You can configure the discovery mode with the method set\_deep\_discovery\_options(NeighborDiscoveryMode, Boolean).

## Configure the deep discovery process

Before discovering the nodes of a network, you can configure the settings of the process. The API provides two methods to configure the discovery timeout and discovery options.

Method	Description
set_deep_discovery_timeouts(Float, Float, Float)	Configures the deep discovery timeouts:  • node_timeout (Float, optional): Maximum duration in seconds of the discovery process used to find neighbors of a node.  • time_bw_requests (Float, optional): Time to wait between node neighbors requests (in seconds)  - For cascade: Time to wait after completion of the a node neighbor discovery process and before next node request.  - For flood: Minimum time to wait between each neighbor request.  • time_bw_scans (Float, optional): Time to wait before starting a new network scan (in seconds)
set_deep_discovery_options(NeighborDiscoveryMode Boolean)	<ul> <li>Configures the deep discovery options:         <ul> <li>deep_mode (NeighborDiscoveryMode, optional): Neighbor discovery mode, the way to perform the network discovery process. See :ref:'deepDiscoveryMode'</li> <li>del_not_discovered_nodes_in_last_scan (Boolean, optional): True to remove nodes from the network if they were not discovered in the last scan.</li> </ul> </li> </ul>

## Configure deep discovery timeout and options

# 2.6.5.3 Standard discovery

This type of discovery process only finds network nodes, it does not include information about the quality of the connections between them.

XBee nodes sleeping may not respond to this request, this means, it may not be found using this discovery process type.

The discovery process runs until the configured timeout expires or, in case of 802.15.4, until the 'end' packet is received (see *Configure the standard discovery process*)

# Configure the standard discovery process

Before discovering the nodes of a network, you can configure the settings of the process. The API provides two methods to configure the discovery timeout and discovery options. These methods set the values in the radio module.

Method	Description
set_discovery_timeout(Float)	Configures the discovery timeout (NT parameter) with
	the given value in seconds.
set_discovery_options(Set <discoveryoptions>)</discoveryoptions>	Configures the discovery options (NO parameter) with the set of options. The set of discovery options contains the different DiscoveryOptions configuration values that are applied to the local XBee module when performing the discovery process. These options are the following:  • DiscoveryOptions.APPEND_DD: Appends the device type identifier (DD) to the information retrieved when a node is discovered. This option is valid for DigiMesh, Point-to-multipoint (Digi Point) and Zigbee protocols.  • DiscoveryOptions.DISCOVER_MYSELF:  The local XBee is returned as a discovered node. This option is valid for all protocols.  • DiscoveryOptions.APPEND_RSSI: Appends the RSSI value of the last hop to the information retrieved when a node is discovered. This option is valid for DigiMesh and Point-to-multipoint (Digi Point) protocols.

# Configure discovery timeout and options

(continues on next page)

[...]

#### 2.6.5.4 Discover the network

The XBeeNetwork object discovery process allows you to discover and store all the XBee nodes that form the network. The XBeeNetwork object provides a method for executing a discovery process of the selected type:

Method	Description
start_discovery_process(Boolean, Integer)	Starts the discovery process, saving the remote XBee
	found inside the XBeeNetwork object.
	<ul> <li>deep (Boolean, optional): True for a deep network scan, False otherwise. See <i>Discovery types</i>.</li> <li>n_deep_scans (Integer, optional): Number of discovery scans to perform. Only for deep discovery.</li> </ul>

When a discovery process has started, you can monitor and manage it using the following methods provided by the XBeeNetwork object:

Method	Description
is_discovery_running()	Returns whether or not the discovery process is running.
stop_discovery_process()	Stops the discovery process that is taking place.

**Warning:** For a standard discovery and depending on your hardware and firmware version, although you call the stop\_discovery\_process method, DigiMesh and DigiPoint modules are blocked until the configured discovery time has elapsed. This means, if you try to get or set any parameter during that time, a TimeoutException may be thrown. This does not occur for:

- XBee 3 modules running DigiMesh firmware 300B or higher.
- XBee SX modules running firmware A008 or higher, 9008 or higher.

Once the process has finished, you can retrieve the list of nodes that form the network using the get\_devices() method provided by the network object. If the discovery process is running, this method returns None.

All discovered XBee nodes are stored in the XBeeNetwork instance.

# Discover the network (deep)

```
# Instantiate a local XBee object.
xbee = XBeeDevice(...)

# Get the XBee network object from the local XBee.
xnet = xbee.get_network()

# Start the discovery process and wait for it to be over.
```

(continues on next page)

```
xnet.start_discovery_process(deep=True, n_deep_scans=1)
while xnet.is_discovery_running():
    time.sleep(0.5)

# Get the list of the nodes in the network.
nodes = xnet.get_devices()

[...]
```

#### Discover the network (standard)

```
# Instantiate a local XBee object.
xbee = XBeeDevice(...)

# Get the XBee network object from the local XBee.
xnet = xbee.get_network()

# Start the discovery process and wait for it to be over.
xnet.start_discovery_process()
while xnet.is_discovery_running():
    time.sleep(0.5)

# Get the list of the nodes in the network.
nodes = xnet.get_devices()

[...]
```

#### Discover the network with an event notification

The API also allows you to add a discovery event listener to notify when:

- New nodes are discovered.
- The process finishes.
- · An error occurs during the process.

## Notify new discovered nodes

To get notifications when nodes are discovered, you must provide a callback before starting the discovery process using the add\_device\_discovered\_callback() method.

# Add a callback to device discovered event

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
# Define the device discovered callback.
def callback(remote):
    [...]
```

(continues on next page)

```
# Get the XBee network object from the local XBee.
xnet = xbee.get_network()

# Add the device discovered callback.
xnet.add_device_discovered_callback(callback)

# Start the discovery process.
xnet.start_discovery_process(deep=True)

[...]
```

Every time a new remote XBee node is discovered all registered device discovered callbacks are executed, even if the discovered node is already in the node list of the network. Each callback receives a RemoteXBeeDevice as argument, with all the available information. Unknown parameters of this remote node are None.

## **Notify discovery finishes**

To get notifications when a discovery process finishes, you must provide a callback before starting the discovery process using the add\_discovery\_process\_finished\_callback() method.

## Add a callback to discovery process finished event

```
# Instantiate a local XBee object.
xbee = XBeeDevice(...)

# Define the discovery process finished callback.
def callback(status):
    if status == NetworkDiscoveryStatus.ERROR_READ_TIMEOUT:
        [...]

# Add the discovery process finished callback.
xnet.add_discovery_process_finished_callback(callback)
[...]
```

When a discovery process finishes (either successfully or with an error), all registered discovery finished callbacks are executed. This method receives a NetworkDiscoveryStatus object as parameter. This status represents the result of the network discovery process.

```
Example: Device discovery
```

The XBee Python Library includes a sample application that displays how to perform a network discovery using a callback. It can be located in the following path:

examples/network/DiscoverDevicesSample/DiscoverDevicesSample.py

# Discover specific nodes

42

The XBeeNetwork object also provides methods to discover specific nodes within a network. This may be useful, for example, if you only need to work with a particular remote node.

Method	Description
dis-	Specify the node identifier of the XBee to find. Returns the remote XBee whose node identifier
cover_device	(Strpings) the one provided or None if the node was not found. In the case of more than one coinci-
	dences, it returns the first one.
dis-	Specify the node identifiers of the XBee nodes to find. Returns a list with the remote XBee nodes
cover_device	s([Mtring]) de identifiers equal those provided.

**Note:** These methods are blocking, so the application will block until the nodes are found or the configured timeout expires.

**Note:** These methods may not discover sleeping nodes.

#### Discover specific nodes

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
[...]
# Get the XBee network object from the local XBee.
xnet = xbee.get_network()
# Discover the remote node whose node ID is 'SOME NODE ID'.
remote = xnet.discover_device("SOME NODE ID")
# Discover the remote nodes whose node IDs are 'ID 2' and 'ID 3'.
remote_list = xnet.discover_devices(["ID 2", "ID 3"])
[...]
```

#### 2.6.5.5 Access discovered nodes

Once a discovery process finishes, the discovered nodes are saved inside the XBeeNetwork object. You can get a list of discovered nodes at any time using the get\_devices().

This is the list of methods provided by the XBeeNetwork object that allow you to retrieve already discovered nodes:

Method	Description
get_devices()	Returns a copy of the list of remote XBee nodes. If any node is added to the network
	after calling this method, the returned list is not updated.
get_device_by_64(XBee6	4RittArthd tress) emote node already in the network whose 64-bit address matches the given
	one or None if the node is not in the network.
get_device_by_16(XBee16RittArds/tess) emote node already in the network whose 16-bit address matches the given	
	one or None if the node is not in the network.
get_device_by_node_id(S	tranger in the remote node already in the network whose node identifier matches the given
	one or None if the node is not in the network.

# Access discovered nodes

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
# Get the XBee network object from the local XBee.
xnet = xbee.get_network()
[...]
x64addr = XBee64BitAddress(...)
node_id = "SOME_XBEE"
# Discover a node based on a 64-bit address.
spec_node = xnet.get_device_by_64(x64addr)
if spec_node is None:
   print("Device with 64-bit addr: %s not found" % str(x64addr))
# Discover a node based on a Node ID.
spec_node = xnet.get_device_by_node_id(node_id)
if spec_node is not None:
   print("Device with node id: %s not found" % node_id)
[...]
```

#### 2.6.5.6 Access connections between nodes

A deep discovery process stores the connections between found nodes inside the XBeeNetwork object. You can get these connections using the get\_connections () method.

This is the list of methods provided by the XBeeNetwork object that allow you to retrieve the connections between nodes:

Method	Description
get_connections()	Returns a copy of the network connections. If any connection is added after the execu-
	tion of this method, returned list is not updated.
get_node_connections(Abstract)&ReeDevice)the connections with the provided node in one of its ends. If any	
	connection is added after the execution of this method, returned list is not updated.

**Warning:** A deep discovery process must be performed to have network connections available.

Each Connection object contains:

- The two nodes between this connection is established.
- The link quality of the connection in both directions (LinkQuality):
  - From node A to node B
  - From node B to node A
- The connection status in both directions (RouteStatus), active, inactive, etc:
  - From node A to node B
  - From node B to node A

#### **Access network connections**

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
# Get the XBee network object from the local XBee.
xnet = xbee.get_network()
[...]
# Start the discovery process and wait for it to be over.
xnet.start_discovery_process(deep=True, n_deep_scans=1)
while xnet.is_discovery_running():
    time.sleep(0.5)
print("%s" % '\n'.join(map(str, xnet.get_connections())))
[...]
```

# 2.6.5.7 Add and remove nodes manually

This section provides information on methods for adding, removing, and clearing the list of remote XBee nodes.

**Note:** These methods modifies the list of nodes inside the XBeeNetwork object, but do not change the real XBee network. They do not trigger a node join event, a disassociation, or a network reset.

# Manually add nodes to the XBee network

There are several methods for adding remote XBee nodes to an XBee network, in addition to the discovery methods provided by the XBeeNetwork object.

Method	Description	
add_ren	_remoter Remoter XB and to the list of remote nodes of the XBeeNetwork object.	
	<b>Notice</b> that this operation does not join the remote XBee to the network; it just adds that node to the list.	
	The node is added to the node list, but may not be physically in the same network.	
	<b>Note</b> that if the given node already exists in the network, it will not be added, but the node in the current	
	network will be updated with the known parameters of the given node.	
	This method returns the same node with its information updated. If the node was not in the list yet, this	
	method returns it without changes.	
add_ren	remotes([RemberXiBeeDxtlee])nodes to add to the list of remote nodes of the XBeeNetwork object.	
	<b>Notice</b> that this operation does not join the remote XBee nodes to the network; it just adds those nodes to	
	the list. Nodes are added to the node list but may not be physically in the same network.	

# Add a remote node manually to the network

```
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
```

2.6. API reference 45

(continues on next page)

```
[...]
# Get the XBee network object from the local XBee.
xnet = xbee.get_network()

# Get the remote XBee node.
remote = xnet.get_remote(...)

# Add the remote node to the network.
xnet.add_remote(remote)

[...]
```

## Remove an existing node from the XBee network

It is also possible to remove a remote XBee from the list of remote XBee nodes of the XBeeNetwork object by calling the following method.

Method	Description	
re-	Specifies the remote XBee to remove from the list of remote nodes of the XBeeNetwork object. If the	
move_devi	cc(RemoteXB ccDevice) in the list, the method will raise a ValueError.	
	<b>Notice</b> that this operation does not disassociates the remote XBee from the actual XBee network; it	
	just deletes the node from the network object list. However, next time you perform a discovery, it	
	could be added again automatically.	

## Remove a remote node from the network

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
[...]
# Get the XBee network object from the local XBee.
xnet = xbee.get_network()
# Get the remote XBee and add it to the network.
remote = xnet.get_remote(...)
xnet.add_remote(remote)
# Remove the remote node from the network.
xnet.remove_device(remote)
[...]
```

## Clear the list of remote XBee nodes from the XBee network

The XBeeNetwork object also includes a method to clear the list of remote nodes. This can be useful when you want to perform a clean discovery, cleaning the list before calling the discovery method.

#### Metho Description

**clear**()Removes all the devices from the list of remote nodes of the network.

**Notice** that this does not imply dismantling the XBee the actual XBee network; it just clears the list of nodes in the XBeeNetwork object. Next time you perform a discovery, the list could be filled with the found remote XBee nodes.

#### Clear the list of remote nodes

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
[...]
# Get the XBee network object from the local XBee.
xnet = xbee.get_network()
# Discover XBee devices in the network and add them to the list of nodes.
[...]
# Clear the list of nodes.
xnet.clear()
[...]
```

#### 2.6.5.8 Listen to network modification events

When a discovery process finds new nodes that were not in the XBee network list (XBeeNetwork or a subclass), they are stored generating a modification event in the XBee network object. A manual removal or addition of an XBee to the network also launches modification events.

The XBee library notifies about these network list modification events to registered callbacks. These events inform about the following network modifications:

- · Addition of new nodes
- Removal of existing nodes
- Update of nodes
- Network clear

To receive any of these modification events you must provide a callback using the add\_network\_modified\_callback() method. This callback must follow the format:

```
def my_callback(event_type, reason, node):
    """
    Callback to notify about a new network modification event.

Args:
    event_type (:class:`.NetworkEventType`): The type of modification.
    reason (:class:`.NetworkEventReason`): The cause of the modification.
    node (:class:`.AbstractXBeeDevice`): The node involved in the
        modification (``None`` for ``NetworkEventType.CLEAR`` events)
    """
[...]
```

When a modification in the network list occurs, all network modification callbacks are executed. Each callback receives the following arguments:

- The type of network modification as a NetworkEventType (addition, removal, update or clear)
- The modification cause as a NetworkEventReason (discovered, discovered as neighbor, received message, hop of a network route, refresh node information, firmware update, manual)
- The XBee node, local or remote, (AbstractXBeeDevice) involved in the modification (None for a clear event type)

## Register a network modifications callback

```
[...]
# Define the network modified callback.
def cb_network_modified(event_type, reason, node):
 print(" >>>> Network event:")
 print("
                 Type: %s (%d)" % (event_type.description, event_type.code))
 print("
                Reason: %s (%d) " % (reason.description, reason.code))
 if not node:
   return
 print("
                Node:")
 print("
                    %s" % node)
xnet = xbee.get_network()
# Add the network modified callback.
xnet.add_network_modified_callback(cb_network_modified)
[...]
```

#### **Network events**

The NetworkEventType class enumerates the possible network cache modification types:

- Addition (NetworkEventType.ADD): A new XBee has just been added to the network cache.
- Deletion (NetworkEventType.DEL): An XBee in the network cache has just been removed.
- Update (NetworkEventType.UPDATE): An existing XBee in the network cache has just been updated. This means any of its parameters (node id, 16-bit address, role, ...) changed.
- Clear (NetworkEventType.CLEAR): The network cached has just been cleared.

As well, NetworkEventReason enumerates the network modification causes:

- NetworkEventReason.DISCOVERED: The node was added/removed/updated during a standard discovery process.
- NetworkEventReason.NEIGHBOR: The node was added/removed/updated during a deep discovery process.
- NetworkEventReason.RECEIVED\_MSG: The node was added/updated after receiving a message from it.
- NetworkEventReason.ROUTE: The node was added/updated as a hop of a received network route.
- NetworkEventReason.READ\_INFO: The node was updated after refreshing its information.

- NetworkEventReason.FIRMWARE\_UPDATE: The node was updated/removed, or the network cleared after a firmware update.
- NetworkEventReason.MANUAL: The node was manually added/updated/removed, or the network cleared.

For example, if, during a deep discovery process, a new node is found and:

- it is not in the network list yet, the addition triggers a new event with:
  - type: NetworkEventType.ADD
  - cause: NetworkEventReason.NEIGHBOR
- it is already in the network list but its node identifier is updated, a new event is raised with:
  - type: NetworkEventType.UPDATE
  - cause: NetworkEventReason.NEIGHBOR
- it is already in the network and nothing has changed, no event is triggered.

## Example: Network modifications

The XBee Python Library includes a sample application that displays how to receive network modification events. It can be located in the following path:

examples/network/NetworkModificationsSample/NetworkModificationsSample.py

## 2.6.6 Communicate with XBee devices

The XBee Python Library provides the ability to communicate with remote nodes in the network, IoT devices and other interfaces of the local device. The communication between XBee devices in a network involves the transmission and reception of data.

**Warning:** Communication features described in this topic and sub-topics are only applicable for local XBee devices. Remote XBee device classes do not include methods for transmitting or receiving data.

## 2.6.6.1 Send and receive data

XBee modules can communicate with other devices that are on the same network and use the same radio frequency. The XBee Python Library provides several methods to send and receive data between the local XBee device and any remote on the network.

- Send data
- · Receive data

#### Send data

A data transmission operation sends data from your local (attached) XBee device to a remote device on the network. The operation sends data in API frames, but the XBee Python library abstracts the process so you only need to specify the device you want to send data to and the data itself.

You can send data either using a unicast or broadcast transmission. Unicast transmissions route data from one source device to one destination device, whereas broadcast transmissions are sent to all devices in the network.

## Send data to one device

Unicast transmissions are sent from one source device to another destination device. The destination device could be an immediate neighbor of the source, or it could be several hops away.

Data transmission can be synchronous or asynchronous, depending on the method used.

# Synchronous operation

This type of operation is blocking. This means the method waits until the transmit status response is received or the default timeout is reached.

The XBeeDevice class of the API provides the following method to perform a synchronous unicast transmission with a remote node of the network:

Method	Description
send_data(RemoteXBeeDevice, String or	Specifies the remote XBee destination object, the data to send and
Bytearray, Integer)	optionally the transmit options.

Protocol-specific classes offer additional synchronous unicast transmission methods apart from the one provided by the XBeeDevice object:

XBee	Method	Description
class		
Zig-	send_data_64_16(XBee64BitAd	d Sepse; ifies the 64-bit and 16-bit destination addresses, the data to send
BeeDe-	XBee16BitAddress, String or	and optionally the transmit options. If you do not know the 16-bit
vice	Bytearray, Integer)	address, use the XBee16BitAddress.UNKNOWN_ADDRESS.
Raw802	Denidedata_16(XBee16BitAddre	ss\$pecifies the 16-bit destination address, the data to send and optionally
	String or Bytearray, Integer)	the transmit options.
	send_data_64(XBee64BitAddre	ss\$pecifies the 64-bit destination address, the data to send and optionally
	String or Bytearray, Integer)	the transmit options.
DigiMe	skend_data_64(XBee64BitAddre	ss\$pecifies the 64-bit destination address, the data to send and optionally
vice	String or Bytearray, Integer)	the transmit options.
Digi-	send_data_64_16(XBee64BitAd	d Seascifies the 64-bit and 16-bit destination addresses, the data to send
Point-	XBee16BitAddress, String or	and optionally the transmit options. If you do not know the 16-bit
De-	Bytearray, Integer)	address, use the XBee16BitAddress.UNKNOWN_ADDRESS.
vice		

# Send data synchronously

The previous methods may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

The default timeout to wait for the send status is two seconds. However, you can configure the timeout using the get\_sync\_ops\_timeout and set\_sync\_ops\_timeout methods of an XBee device class.

## Get/set the timeout for synchronous operations

```
[...]
NEW_TIMEOUT_FOR_SYNC_OPERATIONS = 5 # 5 seconds

device = [...]

# Retrieving the configured timeout for synchronous operations.
print("Current timeout: %d seconds" % device.get_sync_ops_timeout())

[...]

# Configuring the new timeout (in seconds) for synchronous operations.
device.set_sync_ops_timeout(NEW_TIMEOUT_FOR_SYNC_OPERATIONS)

[...]
```

# Example: Synchronous unicast transmission

The XBee Python Library includes a sample application that shows you how to send data to another XBee device on the network. The example is located in the following path:

## examples/communication/SendDataSample

## **Asynchronous operation**

Transmitting data asynchronously means that your application does not block during the transmit process. However, you cannot ensure that the data was successfully sent to the remote device.

The XBeeDevice class of the API provides the following method to perform an asynchronous unicast transmission with a remote node on the network:

Method	Description
send_data_async(RemoteXBeeDevice,	Specifies the remote XBee destination object, the data to send
String or Bytearray, Integer)	and optionally the transmit options.

Protocol-specific classes offer some other asynchronous unicast transmission methods in addition to the one provided by the XBeeDevice object:

XBee	Method	Description
class		
Zig-	send_data_async_64_16(XBee64I	<b>Bis pddiffess</b> the 64-bit and 16-bit destination addresses, the data to send
BeeDe-	XBee16BitAddress, String or	and optionally the transmit options. If you do not know the 16-bit
vice	Bytearray, Integer)	address, use the XBee16BitAddress.UNKNOWN_ADDRESS.
Raw80	2Bendtedata_async_16(XBee16BitA	<b>d8pess</b> fies the 16-bit destination address, the data to send and option-
	String or Bytearray, Integer)	ally the transmit options.
	send_data_async_64(XBee64BitA	<b>d8pess</b> fies the 64-bit destination address, the data to send and option-
	String or Bytearray, Integer)	ally the transmit options.
DigiMe	shend_data_async_64(XBee64BitA	<b>d8pess</b> fies the 64-bit destination address, the data to send and option-
vice	String or Bytearray, Integer)	ally the transmit options.
Digi-	send_data_async_64_16(XBee64I	<b>Bis pddiffess</b> the 64-bit and 16-bit destination addresses, the data to send
Point-	XBee16BitAddress, String or	and optionally the transmit options. If you do not know the 16-bit
De-	Bytearray, Integer)	address, use the XBee16BitAddress.UNKNOWN_ADDRESS.
vice		

## Send data asynchronously

The previous methods may fail for the following reasons:

- All the possible errors are caught as an XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# Example: Asynchronous unicast transmission

The XBee Python Library includes a sample application that shows you how to send data to another XBee device asynchronously. The example is located in the following path:

examples/communication/SendDataAsyncSample

## Send data to all devices of the network

52

Broadcast transmissions are sent from one source device to all the other devices on the network.

All the XBee device classes (generic and protocol specific) provide the same method to send broadcast data:

Method	Description
send_data_broadcast(String or Bytearray, Inte-	Specifies the data to send and optionally the transmit op-
ger)	tions.

#### Send broadcast data

```
[...]
# Instantiate an XBee device object.
device = XBeeDevice("COM1", 9600)
device.open()

# Send broadcast data.
device.send_data_broadcast("Hello XBees!")
[...]
```

The send\_data\_broadcast method may fail for the following reasons:

- Transmit status is not received in the configured timeout, throwing a TimeoutException exception.
- Error types catch as XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The transmit status is not SUCCESS, throwing a TransmitException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# Example: Broadcast transmission

The XBee Python Library includes a sample application that shows you how to send data to all the devices on the network (broadcast). The example is located in the following path:

examples/communication/SendBroadcastDataSample

#### Receive data

The data reception operation allows you to receive and handle data sent by other remote nodes of the network.

There are two different ways to read data from the device:

- **Polling for data**. This mechanism allows you to read (ask) for new data in a polling sequence. The read method blocks until data is received or until a configurable timeout has expired.
- Data reception callback. In this case, you must register a listener that executes a callback each time new data is received by the local XBee device (that is, the device attached to your PC) providing data and other related information.

# Polling for data

The simplest way to read for data is by executing the read\_data method of the local XBee device. This method blocks your application until data from any XBee device of the network is received or the timeout provided has expired:

Method	Description
read_dat	a(Integers) the time to wait for data reception (method blocks during that time and throws a
	TimeoutException if no data is received). If you do not specify a timeout, the method returns
	immediately the read message or None if the device did not receive new data.

## Reading data from any remote XBee device (polling)

```
[...]
# Instantiate an XBee device object.
device = XBeeDevice("COM1", 9600)
device.open()

# Read data.
xbee_message = device.read_data()
[...]
```

The method returns the read data inside an XBeeMessage object. This object contains the following information:

- RemoteXBeeDevice that sent the message.
- Byte array with the contents of the received data.
- Flag indicating if the data was sent via broadcast.
- Time when the message was received.

You can retrieve the previous information using the corresponding attributes of the XBeeMessage object:

## Get the XBeeMessage information

```
[...]
xbee_message = device.read_data()

remote_device = xbee_message.remote_device
data = xbee_message.data
is_broadcast = xbee_message.is_broadcast
timestamp = xbee_message.timestamp
[...]
```

You can also read data from a specific remote XBee device of the network. For that purpose, the XBee device object provides the read\_data\_from method:

Method	Description	
read_data_from (Remittex Bee Denitte, XBee device to read data from and the time to wait for data reception		
Integer)	(method blocks during that time and throws a TimeoutException if no data is received). If	
	you do not specify a timeout, the method returns immediately the read message or None if the	
	device did not receive new data.	

## Read data from a specific remote XBee device (polling)

```
[...]
# Instantiate an XBee device object.
```

(continues on next page)

As in the previous method, this method also returns an XBeeMessage object with all the information inside.

The default timeout to wait for the send status is two seconds. However, you can configure the timeout using the get\_sync\_ops\_timeout and set\_sync\_ops\_timeout methods of an XBee device class.

# Example: Receive data with polling

The XBee Python Library includes a sample application that shows you how to receive data using the polling mechanism. The example is located in the following path:

examples/communication/ReceiveDataPollingSample

## Data reception callback

This mechanism for reading data does not block your application. Instead, you can be notified when new data has been received if you are subscribed or registered to the data reception service using the add\_data\_received\_callback method with a data reception callback as parameter.

## Register for data reception

```
# Instantiate an XBee device object.
device = XBeeDevice("COM1", 9600)
device.open()

# Define callback.
def my_data_received_callback(xbee_message):
    address = xbee_message.remote_device.get_64bit_addr()
    data = xbee_message.data.decode("utf8")
    print("Received data from %s: %s" % (address, data))

# Add the callback.
device.add_data_received_callback(my_data_received_callback)
[...]
```

When new data is received, your callback is executed providing as parameter an XBeeMessage object which contains the data and other useful information:

- RemoteXBeeDevice that sent the message.
- Byte array with the contents of the received data.
- Flag indicating if the data was sent via broadcast.
- Time when the message was received.

To stop listening to new received data, use the del\_data\_received\_callback method to unsubscribe the already-registered callback.

#### **Deregister data reception**

```
def my_data_received_callback(xbee_message):
    [...]

device.add_data_received_callback(my_data_received_callback)
[...]

# Delete the callback
device.del_data_received_callback(my_data_received_callback)
[...]
```

## Example: Register for data reception

The XBee Python Library includes a sample application that shows you how to subscribe to the data reception service to receive data. The example is located in the following path:

examples/communication/ReceiveDataSample

## 2.6.6.2 Send and receive explicit data

Some Zigbee applications may require communication with third-party (non-Digi) RF modules. These applications often send and receive data on different public profiles such as Home Automation or Smart Energy to other modules.

XBee Zigbee modules offer a special type of frame for this purpose. Explicit frames are used to transmit and receive explicit data. When sending public profile packets, the frames transmit the data itself plus the application layer-specific fields—the source and destination endpoints, profile ID, and cluster ID.

**Warning:** Only Zigbee, DigiMesh, and Point-to-Multipoint protocols support the transmission and reception of data in explicit format. This means you cannot transmit or receive explicit data using a generic XBeeDevice object. You must use a protocol-specific XBee device object such as a ZigBeeDevice.

- · Send explicit data
- Receive explicit data

## Send explicit data

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

## Send explicit data to one device

Unicast transmissions are sent from one source device to another destination device. The destination device could be an immediate neighbor of the source, or it could be several hops away.

Unicast explicit data transmission can be a synchronous or asynchronous operation, depending on the method used.

# Synchronous operation

The synchronous data transmission is a blocking operation. That is, the method waits until it either receives the transmit status response or the default timeout is reached.

All local XBee device classes that support explicit data transmission provide a method to transmit unicast and synchronous explicit data to a remote node of the network:

Method	Description
send_expl_data(RemoteXBeeDevice,	Specifies remote XBee destination object, four application layer fields
Integer, Integer, Integer,	(source endpoint, destination endpoint, cluster ID, and profile ID), the
String or Bytearray, Integer)	data to send and optionally the transmit options.

#### Send unicast explicit data synchronously

The previous methods may fail for the following reasons:

- The method throws a TimeoutException exception if the response is not received in the configured timeout.
- Other errors register as XBeeException:
  - If the operating mode of the device is not API or ESCAPED\_API\_MODE, the method throws an InvalidOperatingModeException.
  - If the transmit status is not SUCCESS, the method throws a TransmitException.
  - If there is an error writing to the XBee interface, the method throws a generic XBeeException.

The default timeout to wait for the send status is two seconds. However, you can configure the timeout using the get\_sync\_ops\_timeout and set\_sync\_ops\_timeout methods of an XBee device class.

# Example: Transmit explicit synchronous unicast data

The XBee Python Library includes a sample application that demonstrates how to send explicit data to a remote device of the network (unicast). It can be located in the following path:

examples/communication/explicit/SendExplicitDataSample

#### **Asynchronous operation**

Transmitting explicit data asynchronously means that your application does not block during the transmit process. However, you cannot ensure that the data was successfully sent to the remote device.

All local XBee device classes that support explicit data transmission provide a method to transmit unicast and asynchronous explicit data to a remote node of the network:

Method	Description
send_expl_data_async(RemoteXBeeDevic&pecifies remote XBee destination object, four application layer	
Integer, Integer, Integer, String	(source endpoint, destination endpoint, cluster ID, and profile ID), the
or Bytearray, Integer)	data to send and optionally the transmit options.

## Send unicast explicit data asynchronously

The previous methods may fail for the following reasons:

- All the possible errors are caught as an XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

## Example: Transmit explicit asynchronous unicast data

The XBee Python Library includes a sample application that demonstrates how to send explicit data to other XBee devices asynchronously. It can be located in the following path:

examples/communication/explicit/SendExplicitData AsyncSample

## Send explicit data to all devices in the network

Broadcast transmissions are sent from one source device to all other devices in the network.

All protocol-specific XBee device classes that support the transmission of explicit data provide the same method to send broadcast explicit data:

Method	Description
send_expl_data_broadcast(Integer,	Specifies the four application layer fields (source endpoint, destination
Integer, Integer, String or	endpoint, cluster ID, and profile ID), the data to send and optionally
Bytearray, Integer)	the transmit options.

#### Send broadcast data

```
[...]
# Instantiate a Zigbee device object.
device = ZigBeeDevice("COM1", 9600)
device.open()

# Send broadcast data.
device.send_expl_data_broadcast(0xA0, 0xA1, 0x1554, 0xC105, "Hello XBees!")
[...]
```

The send\_expl\_data\_broadcast method may fail for the following reasons:

- Transmit status is not received in the configured timeout, throwing a TimeoutException exception.
- Error types catch as XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The transmit status is not SUCCESS, throwing a TransmitException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# Example: Send explicit broadcast data

The XBee Python Library includes a sample application that demonstrates how to send explicit data to all devices in the network (broadcast). It can be located in the following path:

examples/communication/explicit/SendBroadcastExplicitDataSample

# Receive explicit data

Some applications developed with the XBee Python Library may require modules to receive data in application layer, or explicit, data format.

To receive data in explicit format, you must first configure the data output mode of the receiver XBee device to explicit format using the set\_api\_output\_mode\_value method.

Method	Description	
get_api_output_mod	eRealure() the API output mode of the data received by the XBee device.	
set_api_output_mode_Spakie(Intleger) I output mode of the data received by the XBee device. Calculate the mode		
	with the method calculate_api_output_mode_value with a set of APIOutputModeBit.	

# Set API output mode

(continues on next page)

Once you have configured the device to receive data in explicit format, you can read it using one of the following mechanisms provided by the XBee device object.

## Polling for explicit data

The simplest way to read for explicit data is by executing the read\_expl\_data method of the local XBee device. This method blocks your application until explicit data from any XBee device of the network is received or the provided timeout has expired:

Method	Description
read_expl_data(Integhr)time to wait in seconds for explicit data reception (method blocks during that time and	
	throws a TimeoutException if no data is received). If you do not specify a timeout, the method
	returns immediately the read message or None if the device did not receive new data.

# Read explicit data from any remote XBee device (polling)

```
# Instantiate a Zigbee device object.
device = ZigBeeDevice("COM1", 9600)
device.open()

# Read data.
xbee_message = device.read_expl_data()
[...]
```

The method returns the read data inside an ExplicitXBeeMessage object. This object contains the following information:

- RemoteXBeeDevice that sent the message.
- Endpoint of the source that initiated the transmission.
- Endpoint of the destination where the message is addressed.
- · Cluster ID where the data was addressed.
- · Profile ID where the data was addressed.
- Byte array with the contents of the received data.
- Flag indicating if the data was sent via broadcast.
- · Time when the message was received.

You can retrieve the previous information using the corresponding attributes of the ExplicitXBeeMessage object:

## Get the ExplicitXBeeMessage information

```
expl_xbee_message = device.read_expl_data()

remote_device = expl_xbee_message.remote_device
source_endpoint = expl_xbee_message.source_endpoint
dest_endpoint = expl_xbee_message.dest_endpoint
cluster_id = expl_xbee_message.cluster_id
profile_id = expl_xbee_message.profile_id
data = xbee_message.data
is_broadcast = expl_xbee_message.is_broadcast
timestamp = expl_xbee_message.timestamp
[...]
```

You can also read explicit data from a specific remote XBee device of the network. For that purpose, the XBee device object provides the read\_expl\_data\_from method:

Method	Description
read_expl_data_fremtteXbnoDeXiBee device to read explicit data from and the time to wait for explicit	
Integer)	data reception (method blocks during that time and throws a TimeoutException if no data
	is received). If you do not specify a timeout, the method returns immediately the read message
	or None if the device did not receive new data.

# Read explicit data from a specific remote XBee device (polling)

As in the previous method, this method also returns an ExplicitXBeeMessage object with all the information inside.

The default timeout to wait for data is two seconds. However, you can configure the timeout using the get\_sync\_ops\_timeout and set\_sync\_ops\_timeout methods of an XBee device class.

Example: Receive explicit data with polling

The XBee Python Library includes a sample application that demonstrates how to receive explicit data using the polling mechanism. It can be located in the following path:

examples/communication/explicit/ReceiveExplicitDataPollingSample

## **Explicit data reception callback**

This mechanism for reading explicit data does not block your application. Instead, you can be notified when new explicit data has been received if you are subscribed or registered to the explicit data reception service by using the add\_expl\_data\_received\_callback.

# **Explicit data reception registration**

```
[...]
# Instantiate a Zigbee device object.
device = ZigBeeDevice("COM1", 9600)
device.open()
# Define callback.
def my_expl_data_received_callback(expl_xbee_message):
   address = expl_xbee_message.remote_device.get_64bit_addr()
   source_endpoint = expl_xbee_message.source_endpoint
   dest_endpoint = expl_xbee_message.dest_endpoint
   cluster = expl_xbee_message.cluster_id
    profile = expl_xbee_message.profile_id
   data = expl_xbee_message.data.decode("utf8")
   print("Received explicit data from %s: %s" % (address, data))
# Add the callback.
device.add_expl_data_received_callback(my_expl_data_received_callback)
[\ldots]
```

When new explicit data is received, your callback is executed providing as parameter an ExplicitXBeeMessage object which contains the data and other useful information:

- RemoteXBeeDevice that sent the message.
- Endpoint of the source that initiated the transmission.
- Endpoint of the destination where the message is addressed.
- · Cluster ID where the data was addressed.
- · Profile ID where the data was addressed.
- Byte array with the contents of the received data.
- Flag indicating if the data was sent via broadcast.
- Time when the message was received.

To stop listening to new received explicit data, use the del\_expl\_data\_received\_callback method to unsubscribe the already-registered callback.

## **Explicit data reception deregistration**

```
def my_expl_data_received_callback(xbee_message):
    [...]
device.add_expl_data_received_callback(my_expl_data_received_callback)
```

(continues on next page)

```
# Delete the callback
device.del_expl_data_received_callback(my_expl_data_received_callback)
[...]
```

#### Example: Receive explicit data via callback

The XBee Python Library includes a sample application that demonstrates how to subscribe to the explicit data reception service in order to receive explicit data. It can be located in the following path:

examples/communication/explicit/ReceiveExplicitDataSample

**Note:** If your XBee module is configured to receive explicit data (API output mode greater than 0) and another device sends non-explicit data or a IO sample, you receive an explicit message whose application layer field values are:

• For remote data:

- Source endpoint: 0xE8

- Destination endpoint: 0xE8

Cluster ID: 0x0011Profile ID: 0xC105

• For remote IO sample:

- Source endpoint: 0xE8

- Destination endpoint: 0xE8

Cluster ID: 0x0092Profile ID: 0xC105

That is, when an XBee receives explicit data with these values, the message notifies the following reception callbacks in case you have registered them:

- Explicit and non-explicit data callbacks when receiving remote data.
- Explicit data callback and IO sample callback when receiving remote samples.

If you read the received data with the polling mechanism, you also receive the message through both methods.

#### 2.6.6.3 Send and receive IP data

In contrast to XBee protocols like Zigbee, DigiMesh or 802.15.4, where the devices are connected each other, in cellular and Wi-Fi protocols the modules are part of the Internet.

XBee Cellular and Wi-Fi modules offer a special type of frame for communicating with other Internet-connected devices. It allows sending and receiving data specifying the destination IP address, port, and protocol (TCP, TCP SSL or UDP).

Warning: Only Cellular and Wi-Fi protocols support the transmission and reception of IP data. This means you cannot transmit or receive IP data using a generic XBeeDevice object; you must use the protocol-specific XBee device objects CellularDevice or WiFiDevice.

- · Send IP data
- · Receive IP data

#### Send IP data

IP data transmission can be a synchronous or asynchronous operation, depending on the method you use.

# Synchronous operation

The synchronous data transmission is a blocking operation; that is, the method waits until it either receives the transmit status response or it reaches the default timeout.

The CellularDevice and WiFiDevice classes include several methods to transmit IP data synchronously:

Method	Description
send_ip_data(IPv4Address,	Specifies the destination IP address, destination port, IP protocol (UDP, TCP
Integer, IPProtocol, String or	or TCP SSL), data to send for transmissions and whether the socket should be
Bytearray, Boolean)	closed after the transmission or not (optional).

## Send network data synchronously

```
# Instantiate a Cellular device object.
xbee = CellularDevice("COM1", 9600)
xbee.open()

# Send IP data using TCP.
dest_addr = IPv4Address("56.23.102.96")
dest_port = 5050
protocol = IPProtocol.TCP
data = "Hello XBee!"

xbee.send_ip_data(dest_addr, dest_port, protocol, data)
[...]
```

The send\_ip\_data method may fail for the following reasons:

- There is a timeout setting the IP addressing parameter, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

## Example: Transmit IP data synchronously

The XBee Python Library includes a sample application that demonstrates how to send IP data. You can locate the example in the following path:

examples/communication/ip/SendIPDataSample

## Example: Transmit UDP data

The XBee Python Library includes a sample application that demonstrates how to send UDP data. You can locate the example in the following path:

examples/communication/ip/SendUDPDataSample

# Example: Connect to echo server

The XBee Python Library includes a sample application that demonstrates how to connect to an echo server, send a message to it and receive its response. You can locate the example in the following path:

examples/communication/ip/ConnectToEchoServerSample

# **Asynchronous operation**

Transmitting IP data asynchronously means that your application does not block during the transmit process. However, you cannot ensure that the data was successfully sent.

The CellularDevice and WiFiDevice classes include several methods to transmit IP data asynchronously:

Method	Description
send_ip_data_async(IPv4Address	, Specifies the destination IP address, destination port, IP protocol (UDP, TCP
Integer, IPProtocol, String or	or TCP SSL), data to send for transmissions and whether the socket should
Bytearray, Boolean)	be closed after the transmission or not (optional).

## Send network data asynchronously

```
# Instantiate a Cellular device object.
xbee = CellularDevice("COM1", 9600)
xbee.open()

# Send IP data using TCP.
dest_addr = IPv4Address("56.23.102.96")
dest_port = 5050
protocol = IPProtocol.TCP
data = "Hello XBee!"

xbee.send_ip_data_async(dest_addr, dest_port, protocol, data)
[...]
```

The send\_ip\_data\_async method may fail for the following reasons:

- All possible errors are caught as XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

#### Receive IP data

Some applications developed with the XBee Python Library may require modules to receive IP data.

XBee Cellular and Wi-Fi modules operate the same way as other TCP/IP devices. They can initiate communications with other devices or listen for TCP or UDP transmissions at a specific port. In either case, you must apply any of the receive methods explained in this section in order to read IP data from other devices.

# Listen for incoming transmissions

If the cellular or Wi-Fi module operates as a server, listening for incoming TCP or UDP transmissions, you must start listening at a specific port, similar to the bind operation of a socket. The XBee Python Library provides a method to listen for incoming transmissions:

Method	Description
start_listening(Integer)	Starts listening for incoming IP transmissions in the provided port.

## Listen for incoming transmissions

```
# Instantiate a Cellular device object.
device = CellularDevice("COM1", 9600)
device.open()
# Listen for TCP or UDP transmissions at port 1234.
device.start_listening(1234);
[...]
```

The start\_listening method may fail for the following reasons:

- If the listening port provided is lesser than 0 or greater than 65535, the method throws a ValueError error.
- If there is a timeout setting the listening port, the method throws a TimeoutException exception.
- Errors that register as an XBeeException:
  - If the operating mode of the device is not API or ESCAPED\_API\_MODE, the method throws an InvalidOperatingModeException.
  - If the response of the listening port command is not valid, the method throws an ATCommandException.
  - If there is an error writing to the XBee interface, the method throws a generic XBeeException.

You can call the stop\_listening method to stop listening for incoming TCP or UDP transmissions:

Method	Description
stop_listening()	Stops listening for incoming IP transmissions.

#### Stop listening for incoming transmissions

```
[...]
# Instantiate a Cellular device object.
device = CellularDevice("COM1", 9600)
device.open()
```

(continues on next page)

```
# Stop listening for TCP or UDP transmissions.
device.stop_listening()
[...]
```

The stop\_listening method may fail for the following reasons:

- There is a timeout setting the listening port, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# Polling for IP data

The simplest way to read IP data is by executing the read\_ip\_data method of the local Cellular or Wi-Fi devices. This method blocks your application until IP data is received or the provided timeout has expired.

Method	Description
read_ip_da	<b>taSpeteiges</b> ) the time to wait in seconds for IP data reception (method blocks during that time or until
	IP data is received). If you don't specify a timeout, the method uses the default receive timeout
	configured in XBeeDevice.

# Read IP data (polling)

```
# Instantiate a Cellular device object.
device = CellularDevice("COM1", 9600)
device.open()

# Read IP data.
ip_message = device.read_ip_data()

[...]
```

The method returns the read data inside an IPMessage object and contains the following information:

- IP address of the device that sent the data
- · Transmission protocol
- Source and destination ports
- Byte array with the contents of the received data

You can retrieve the previous information using the corresponding attributes of the IPMessage object:

# Get the IPMessage information

```
[...]
# Instantiate a cellular device object.
(continues on next page)
```

```
device = CellularDevice("COM1", 9600)
device.open()

# Read IP data.
ip_message = device.read_ip_data()

ip_addr = ip_message.ip_addr
source_port = ip_message.source_port
dest_port = ip_message.dest_port
protocol = ip_message.protocol
data = ip_message.data

[...]
```

You can also read IP data that comes from a specific IP address. For that purpose, the cellular and Wi-Fi device objects provide the read\_ip\_data\_from method:

# Read IP data from a specific IP address (polling)

```
[...]
# Instantiate a cellular device object.
device = CellularDevice("COM1", 9600)
device.open()

# Read IP data.
ip_message = device.read_ip_data_from(IPv4Address("52.36.102.96"))
[...]
```

This method also returns an IPMessage object containing the same information described before.

### Example: Receive IP data with polling

The XBee Python Library includes a sample application that demonstrates how to receive IP data using the polling mechanism. You can locate the example in the following path:

examples/communication/ip/ConnectToEchoServerSample

# IP data reception callback

This mechanism for reading IP data does not block your application. Instead, you can be notified when new IP data has been received if you have subscribed or registered with the IP data reception service by using the add\_ip\_data\_received\_callback method.

### IP data reception registration

```
[...]
# Instantiate a Cellular device object.
device = CellularDevice("COM1", 9600)
device.open()
# Define the callback.
```

(continues on next page)

```
def my_ip_data_received_callback(ip_message):
    print("Received IP data from %s: %s" % (ip_message.ip_addr, ip_message.data))

# Add the callback.
device.add_ip_data_received_callback(my_ip_data_received_callback)

[...]
```

When new IP data is received, your callback is executed providing as parameter an IPMessage object which contains the data and other useful information:

- IP address of the device that sent the data
- Transmission protocol
- · Source and destination ports
- Byte array with the contents of the received data

To stop listening to new received IP data, use the del\_ip\_data\_received\_callback method to unsubscribe the already-registered listener.

# Data reception deregistration

```
[...]
device = [...]

def my_ip_data_received_callback(ip_message):
        [...]

device.add_ip_data_received_callback(my_ip_data_received_callback)
[...]

# Delete the IP data callback.
device.del_ip_data_received_callback(my_ip_data_received_callback)
[...]
```

#### Example: Receive IP data with listener

The XBee Python Library includes a sample application that demonstrates how to receive IP data using the listener. You can locate the example in the following path:

examples/communication/ip/ReceiveIPDataSample

# 2.6.6.4 Send and receive SMS messages

Another feature of the XBee Cellular module is the ability to send and receive Short Message Service (SMS) transmissions. This allows you to send and receive text messages to and from an SMS capable device such as a mobile phone.

For that purpose, these modules offer a special type of frame for sending and receiving text messages, specifying the destination phone number and data.

**Warning:** Only Cellular protocol supports the transmission and reception of SMS. This means you cannot send or receive text messages using a generic XBeeDevice object; you must use the protocol-specific XBee device object CellularDevice.

- Send SMS messages
- Receive SMS messages

## Send SMS messages

SMS transmissions can be a synchronous or asynchronous operation, depending on the method you use.

# Synchronous operation

The synchronous SMS transmission is a blocking operation; that is, the method waits until it either receives the transmit status response or it reaches the default timeout.

The CellularDevice class includes the following method to send SMS messages synchronously:

Method	Description
send_sms(String,	Specifies the the phone number to send the SMS to and the data to send as the body of
String)	the SMS message.

# Send SMS message synchronously

```
[...]
# Instantiate a Cellular device object.
xbee = CellularDevice("COM1", 9600)
xbee.open()

phone_number = "+34665963205"
data = "Hello XBee!"

# Send SMS message.
xbee.send_sms(phone_number, data)
[...]
```

The send\_sms method may fail for the following reasons:

- If the response is not received in the configured timeout, the method throws a TimeoutException.
- If the phone number has an invalid format, the method throws a ValueError.
- Errors register as XBeeException:
  - If the operating mode of the device is not API or ESCAPED\_API\_MODE, the method throws an InvalidOperatingModeException.
  - If there is an error writing to the XBee interface, the method throws a generic XBeeException.

```
Example: Send synchronous SMS
```

The XBee Python Library includes a sample application that demonstrates how to send SMS messages. You can locate the example in the following path:

examples/communication/cellular/SendSMSSample

# **Asynchronous operation**

Transmitting SMS messages asynchronously means that your application does not block during the transmit process. However, you cannot verify the SMS was successfully sent.

The CellularDevice class includes the following method to send SMS asynchronously:

Method	Description
send_sms_async(String,	Specifies the the phone number to send the SMS to and the data to send as the body
String)	of the SMS message.

### Send SMS message asynchronously

```
[...]
# Instantiate a Cellular device object.
xbee = CellularDevice("COM1", 9600)
xbee.open()

phone_number = "+34665963205"
data = "Hello XBee!"

# Send SMS message.
xbee.send_sms_async(phone_number, data)
[...]
```

The send\_sms\_async method may fail for the following reasons:

- If the phone number has an invalid format, the method throws a ValueError.
- Errors register as XBeeException:
  - If the operating mode of the device is not API or ESCAPED\_API\_MODE, the method throws an InvalidOperatingModeException.
  - If there is an error writing to the XBee interface, the method throws a generic XBeeException.

# **Receive SMS messages**

Some applications developed with the XBee Python Library may require modules to receive SMS messages.

### SMS reception callback

You can be notified when a new SMS has been received if you are subscribed or registered to the SMS reception service by using the add\_sms\_callback method.

## SMS reception registration

```
# Instantiate a cellular device object.
device = CellularDevice("COM1", 9600)
device.open()

# Define the callback.
def my_sms_callback(sms_message):
    print("Received SMS from %s: %s" % (sms_message.phone_number, sms_message.data))

# Add the callback.
device.add_sms_callback(my_sms_callback)
[...]
```

When a new SMS message is received, your callback is executed providing an SMSMessage object as paramater. This object contains the data and the phone number that sent the message.

To stop listening to new SMS messages, use the del\_sms\_callback method to unsubscribe the already-registered listener.

## **Deregister SMS reception**

```
[...]
device = [...]

def my_sms_callback(sms_message):
    [...]

device.add_sms_callback(my_sms_callback)

[...]

# Delete the SMS callback.
device.del_sms_callback(my_sms_callback)

[...]
```

# Example: Receive SMS messages

The XBee Python Library includes a sample application that demonstrates how to subscribe to the SMS reception service in order to receive text messages. You can locate the example in the following path:

examples/communication/cellular/ReceiveSMSSample

# 2.6.6.5 Send and receive Bluetooth data

XBee3 modules have the ability to send and receive data from the Bluetooth Low Energy interface of the local XBee device through User Data Relay frames. This can be useful if your application wants to transmit or receive data from a cellphone connected to it over BLE.

**Warning:** Only XBee3 modules support Bluetooth Low Energy. This means that you cannot transmit or receive Bluetooth data if you don't have one of these modules.

- · Send Bluetooth data
- · Receive Bluetooth data

#### Send Bluetooth data

The XBeeDevice class and its subclasses provide the following method to send data to the Bluetooth Low Energy interface:

Method	Description
send_bluetooth_data(Bytearray)	Specifies the data to send to the Bluetooth Low Energy interface.

This method is asynchronous, which means that your application does not block during the transmit process.

#### Send data to Bluetooth

```
# Instantiate an XBee device object.
device = XBeeDevice("COM1", 9600)
device.open()
data = "Bluetooth, are you there?"
# Send the data to the Bluetooth interface.
device.send_bluetooth_data(data.encode("utf8"))
[...]
```

The send\_bluetooth\_data method may fail for the following reasons:

- Errors register as XBeeException:
  - If the operating mode of the device is not API or ESCAPED\_API\_MODE, the method throws an InvalidOperatingModeException.
  - If there is an error writing to the XBee interface, the method throws a generic XBeeException.

## Example: Send Bluetooth data

The XBee Python Library includes a sample application that demonstrates how to send data to the Bluetooth interface. You can locate the example in the following path:

examples/communication/bluetooth/SendBluetoothDataSample

## **Receive Bluetooth data**

You can be notified when new data from the Bluetooth Low Energy interface has been received if you are subscribed or registered to the Bluetooth data reception service by using the add\_bluetooth\_data\_received\_callback method.

### Bluetooth data reception registration

```
[...]
# Instantiate an XBee device object.
device = XBeeDevice("COM1", 9600)
```

2.6. API reference 73

(continues on next page)

```
device.open()

# Define the callback.
def my_bluetooth_data_callback(data):
    print("Data received from the Bluetooth interface >> '%s'" % data.decode("utf-8"))

# Add the callback.
device.add_bluetooth_data_received_callback(my_bluetooth_data_callback)

[...]
```

When a new data from the Bluetooth interface is received, your callback is executed providing the data in byte array format as parameter.

To stop listening to new data messages from the Bluetooth interface, use the del\_bluetooth\_data\_received\_callback method to unsubscribe the already-registered listener.

### **Deregister Bluetooth data reception**

```
[...]
device = [...]

def my_bluetooth_data_callback(data):
       [...]

device.add_bluetooth_data_received_callback(my_bluetooth_data_callback)
[...]

# Delete the Bluetooth data callback.
device.del_bluetooth_data_received_callback(my_bluetooth_data_callback)
[...]
```

# Example: Receive Bluetooth data

The XBee Python Library includes a sample application that demonstrates how to subscribe to the Bluetooth data reception service in order to receive data from the Bluetooth Low Energy interface. You can locate the example in the following path:

examples/communication/bluetooth/ReceiveBluetoothDataSample

## 2.6.6.6 Send and receive MicroPython data

XBee3 modules have the ability to send and receive data from the MicroPython interface of the local XBee device through User Data Relay frames. This can be useful if your application wants to transmit or receive data from a MicroPython program running on the module.

**Warning:** Only XBee3 and XBee Cellular modules support MicroPython. This means that you cannot transmit or receive MicroPython data if you don't have one of these modules.

- Send MicroPython data
- Receive MicroPython data

# Send MicroPython data

The XBeeDevice class and its subclasses provide the following method to send data to the MicroPython interface:

Method	Description
send_micropython_data(Bytearray)	Specifies the data to send to the MicroPython interface.

This method is asynchronous, which means that your application does not block during the transmit process.

#### Send data to MicroPython

```
# Instantiate an XBee device object.
device = XBeeDevice("COM1", 9600)
device.open()

data = "MicroPython, are you there?"

# Send the data to the MicroPython interface.
device.send_micropython_data(data.encode("utf8"))

[...]
```

The send\_micropython\_data method may fail for the following reasons:

- Errors register as XBeeException:
  - If the operating mode of the device is not API or ESCAPED\_API\_MODE, the method throws an InvalidOperatingModeException.
  - If there is an error writing to the XBee interface, the method throws a generic XBeeException.

# Example: Send MicroPython data

The XBee Python Library includes a sample application that demonstrates how to send data to the MicroPython interface. You can locate the example in the following path:

examples/communication/micropython/SendMicroPythonDataSample

### Receive MicroPython data

You can be notified when new data from the MicroPython interface has been received if you are subscribed or registered to the MicroPython data reception service by using the add\_micropython\_data\_received\_callback method.

# MicroPython data reception registration

(continues on next page)

```
# Add the callback.
device.add_micropython_data_received_callback(my_micropython_data_callback)
[...]
```

When a new data from the MicroPython interface is received, your callback is executed providing the data in byte array format as parameter.

To stop listening to new data messages from the MicroPython interface, use the del\_micropython\_data\_received\_callback method to unsubscribe the already-registered listener.

# **Deregister MicroPython data reception**

```
[...]
device = [...]

def my_micropython_data_callback(data):
        [...]

device.add_micropython_data_received_callback(my_micropython_data_callback)
[...]

# Delete the MicroPython data callback.
device.del_micropython_data_received_callback(my_micropython_data_callback)
[...]
```

# Example: Receive MicroPython data

The XBee Python Library includes a sample application that demonstrates how to subscribe to the MicroPython data reception service in order to receive data from the MicroPython interface. You can locate the example in the following path:

examples/communication/micropython/ReceiveMicroPythonDataSample

## 2.6.6.7 Receive modem status events

A local XBee device is able to determine when it connects to a network, when it is disconnected, and when any kind of error or other events occur. The local device generates these events, and they can be handled using the XBee Python library via the modem status frames reception.

When a modem status frame is received, you are notified through the callback of a custom listener so you can take the proper actions depending on the event received.

For that purpose, you must subscribe or register to the modem status reception service using a modem status listener as parameter with the method add\_modem\_status\_received\_callback.

## Subscribe to modem status reception service

```
[...]
# Instantiate an XBee device object.
device = XBeeDevice("COM1", 9600)
device.open()
```

(continues on next page)

```
# Define the callback.
def my_modem_status_callback(status):
    print("Modem status: %s" % status.description)

# Add the callback.
device.add_modem_status_received_callback(my_modem_status_callback)

[...]
```

When a new modern status is received, your callback is executed providing as parameter a ModernStatus object.

To stop listening to new modem statuses, use the del\_modem\_status\_received\_callback method to unsubscribe the already-registered listener.

## **Deregister modem status**

```
[...]
device = [...]

def my_modem_status_callback(status):
    [...]

device.add_modem_status_received_callback(my_modem_status_callback)
[...]

# Delete the modem status callback.
device.del_modem_status_received_callback(my_modem_status_callback)
[...]
```

#### Example: Subscribe to modem status reception service

The XBee Python Library includes a sample application that shows you how to subscribe to the modem status reception service to receive modem status events. The example is located in the following path: examples/communication/ReceiveModemStatusSample

## 2.6.6.8 Communicate using XBee sockets

Starting from firmware versions \*13, the XBee Cellular product line includes a new set of frames to communicate with other Internet-connected devices using sockets.

The XBee Python Library provides several methods that allow you to create, connect, bind and close a socket, as well as send and receive data with it. You can use this API where the existing methods listed in the *Send and receive IP data* section limit the possibilities for an application.

**Warning:** Only the Cellular protocol supports the use of XBee sockets. This means you cannot use this API with a generic XBeeDevice object; you must use the protocol-specific XBee device object CellularDevice.

The XBee socket API is available through the socket class of the digi.xbee.xsocket module.

### Create an XBee socket

Before working with an XBee socket to communicate with other devices, you have to instantiate a socket object in order to create it. To do so, you need to provide the following parameters:

- XBee Cellular device object used to work with the socket.
- IP protocol of the socket (optional). It can be IPProtocol.TCP (default), IPProtocol.UDP or IPProtocol.TCP\_SSL.

#### Create an XBee socket

```
from digi.xbee import xsocket
from digi.xbee.devices import CellularDevice
from digi.xbee.models.protocol import IPProtocol

# Create and open an XBee Cellular device.
device = CellularDevice("COM1", 9600)
device.open()

# Create a new XBee socket.
sock = xsocket.socket(device, IPProtocol.TCP)
```

#### Work with an XBee socket

78

Once the XBee socket is created, you can work with it to behave as a client or a server. The API offers the following methods:

Method	Description
con-	Connects to a remote socket at the provided address. The address must be a pair (host, port),
nect(Tuple)	where <i>host</i> is the domain name or string representation of an IPv4 and <i>port</i> is the numeric port value.
close()	Closes the socket.
bind(Tuple)	Binds the socket to the provided address. The address must be a pair (host, port), where host
	is the local interface (not used) and <i>port</i> is the numeric port value. The socket must not already be
	bound.
lis-	Enables a server to accept connections.
ten(Integer)	
accept()	Accepts a connection. The socket must be bound to an address and listening for connections. The
	return value is a pair (conn, address) where conn is a new socket object usable to send and
	receive data on the connection, and address is a pair (host, port) with the address bound to
	the socket on the other end of the connection.
send(Bytear	asynds the provided data to the socket. The socket must be connected to a remote socket.
	<b>rsay</b> ds the provided data to the socket. The socket should not be connected to a remote socket, since
Tuple)	the destination socket is specified by address (a pair (host, port)).
recv(Integer	Receives data from the socket, specifying the maximum amount of data to be received at once. The
	return value is a bytearray object representing the data received.
recvfrom(In	<b>reger</b> vives data from the socket, specifying the maximum amount of data to be received at once. The
	return value is a pair (bytes, address) where bytes is a bytearray object representing the data
	received and address is the address of the socket sending the data(a pair (host, port)).
getsock-	Returns the value of the provided socket option.
opt(SocketO	
setsock-	Sets the value of the provided socket option.
opt(SocketO	ption,
Bytear-	
ray)	
gettime-	Returns the configured socket timeout in seconds.
out()	
settime-	Sets the socket timeout in seconds.
out(Integer)	
getblock-	Returns whether the socket is in blocking mode or not.
ing()	
setblock-	Sets the socket in blocking or non-blocking mode. In blocking mode, operations block until com-
ing(Boolean)	• • • • • • • • • • • • • • • • • • • •
	pleted within the configured timeout.
get_sock_inf	one turns the information of the socket, including the socket ID, state, protocol, local port, remote
odd godrat	port and remote address.
	stated stall bands (Functillh) ack to be notified when a new socket state is received.
dei_socket_s	tabeleathback(Bunktion)cket state callback.

# **Client sockets**

When the socket acts as a client, you just have to create and connect the socket before sending or receiving data with a remote host.

# Work with an XBee socket as client

```
[...]

HOST = "numbersapi.com"

PORT = "80"

(continues on next page)
```

```
REQUEST = "GET /random/trivia HTTP/1.1\r\nHost: numbersapi.com\r\n\r\n"

# Create and open an XBee Cellular device.
device = CellularDevice("COM1", 9600)
device.open()

# Create a new XBee socket.
with xsocket.socket(device, IPProtocol.TCP) as sock:
    # Connect the socket.
    sock.connect((HOST, PORT))

# Send an HTTP request.
    sock.send(REQUEST.encode("utf8"))

# Receive and print the response.
    data = sock.recv(1024)
    print(data.decode("utf8"))
```

# Example: Create a TCP client socket

The XBee Python Library includes a sample application that shows you how to create a TCP client socket to send HTTP requests. The example is located in the following path:

examples/communication/socket/SocketTCPClientSample

#### Server sockets

When the socket acts as a server, you must create the socket and then perform the sequence bind(), listen(), accept().

### Work with an XBee socket as server

```
[...]
PORT = "1234"
# Create and open an XBee Cellular device.
device = CellularDevice("COM1", 9600)
device.open()
# Create a new XBee socket.
with xsocket.socket(device, IPProtocol.TCP) as sock:
    # Bind the socket to the local port.
   sock.bind((None, PORT))
    # Listen for new connections.
   sock.listen()
    # Accept new connections.
   conn, addr = sock.accept()
   with conn:
        print ("Connected by %s", str(addr))
        while True:
            # Print the received data (if any).
            data = conn.recv(1024)
```

(continues on next page)

```
if data:
    print(data.decode("utf8"))
```

### Example: Create a TCP server socket

The XBee Python Library includes a sample application that shows you how to create a TCP server socket to receive data from incoming sockets. The example is located in the following path:

examples/communication/socket/SocketTCPServerSample

# Example: Create a UDP server/client socket

The XBee Python Library includes a sample application that shows how to create a UDP socket to deliver messages to a server and listen for data coming from multiple peers. The example is located in the following path:

examples/communication/socket/SocketUDPServerClientSample

# 2.6.7 Handle analog and digital IO lines

All the XBee modules, regardless of the protocol they run, have a set of IO lines (pins). You can use these pins to connect sensors or actuators and configure them with specific behavior.

You can configure the IO lines of an XBee device to be digital input/output (DIO), analog to digital converter (ADC), or pulse-width modulation output (PWM). The configuration you provide to a line depends on the device where you want to connect.

**Note:** All the IO management features displayed in this topic and sub-topics are applicable for both local and remote XBee devices.

The XBee Python Library exposes an easy way to configure, read, and write the IO lines of the local and remote XBee devices through the following corresponding classes:

- XBeeDevice for local devices.
- RemoteXBeeDevice for remotes.

# 2.6.7.1 Configure the IO lines

All XBee device objects include a configuration method, set\_io\_configuration(), where you can specify the IO line being configured and the desired function being set.

For the IO line parameter, the API provides an enumerator called IOLine that helps you specify the desired IO line easily by functional name. This enumerator is used along all the IO related methods in the API.

The supported functions are also contained in an enumerator called IOMode. You can choose between the following functions:

- DISABLED
- SPECIAL\_FUNCTIONALITY (Shouldn't be used to configure IOs)
- PWM
- ADC
- DIGITAL\_IN
- DIGITAL\_OUT\_LOW

• DIGITAL OUT HIGH

#### Configure local or remote IO lines

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
# Instantiate a remote XBee device object.
remote_xbee = RemoteXBeeDevice(local_xbee, XBee64BitAddress.from_hex_string(
→"0013A20012345678"))
# Configure the DIO1_AD1 line of the local device to be Digital output (set high by.
\rightarrow default).
local_xbee.set_io_configuration(IOLine.DIO1_AD1, IOMode.DIGITAL_OUT_HIGH)
# Configure the DIO2_AD2 line of the local device to be Digital input.
local_xbee.set_io_configuration(IOLine.DIO2_AD2, IOMode.DIGITAL_IN)
# Configure the DIO3_AD3 line of the remote device to be Analog input (ADC).
remote_xbee.set_io_configuration(IOLine.DIO3_AD3, IOMode.ADC)
# Configure the DIO10_PWM0 line of the remote device to be PWM output (PWM).
remote_xbee.set_io_configuration(IOLine.DIO10_PWM0, IOMode.PWM)
[...]
```

The set io configuration () method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

You can read the current configuration of any IO line the same way an IO line can be configured with a desired function using the corresponding getter, get\_io\_configuration().

# **Get IO configuration**

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Get the configuration mode of the DIO1_AD1 line.
io_mode = local_xbee.get_io_configuration(IOLine.DIO1_AD1)
[...]
```

The get\_io\_configuration() method may fail for the following reasons:

• ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.

- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# **Digital Input/Output**

If your IO line is configured as digital output, you can set its state (high/low) easily. All the XBee device classes provide the method, set\_dio\_value(), with the desired IOLine as the first parameter and an IOValue as the second. The IOValue enumerator includes HIGH and LOW as possible values.

### Set digital output values

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

# Set the DIO2_AD2 line low.
local_xbee.set_dio_value(IOLine.DIO2_AD2, IOValue.LOW)

# Set the DIO2_AD2 line high.
local_xbee.set_dio_value(IOLine.DIO2_AD2, IOValue.HIGH)

[...]
```

The set\_dio\_value() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

You can also read the current status of the pin (high/low) by issuing the method get\_dio\_value(). The parameter of the method must be the IO line to be read.

# Read digital input values

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
# Get the value of the DIO2_AD2.
value = local_xbee.get_dio_value(IOLine.DIO2_AD2)
[...]
```

The get\_dio\_value() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - If the received response does not contain the value for the given IO line, throwing an OperationNotSupportedException. This can happen (for example) if you try to read the DIO value of an IO line that is not configured as DIO.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

### Example: Handle DIO IO lines

The XBee Python Library includes two sample applications that demonstrate how to handle DIO lines in your local and remote XBee Devices. The examples are located in the following path:

examples/io/LocalDIOSample/LocalDIOSample.py examples/io/RemoteDIOSample/RemoteDIOSample.py

#### **ADC**

When you configure an IO line as analog to digital converter (ADC), you can only read its value (counts) with get\_adc\_value(). In this case, the method used to read ADCs is different than the digital I/O method, but the parameter provided is the same: the IO line to read the value from.

#### Read ADC values

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
# Get the value of the DIO 3 (analog to digital converter).
value = local_xbee.get_adc_value(IOLine.DIO3_AD3)
[...]
```

The get\_adc\_value() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - If the received response does not contain the value for the given IO line, throwing an OperationNotSupportedException. This can happen (for example) if you try to read the ADC value of an IO line that is not configured as ADC.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

```
Example: Handle ADC IO lines
```

The XBee Python Library includes two sample applications that demonstrate how to handle ADC lines in your local and remote XBee devices. The examples are located in the following path:

examples/io/LocalADCSample/LocalADCSample.py examples/io/RemoteADCSample/RemoteADCSample.py

#### **PWM**

Not all the XBee protocols support pulse-width modulation (PWM) output handling, but the XBee Python Library provides functionality to manage them. When you configure an IO line as PWM output, you must use specific methods to set and read the duty cycle of the PWM.

For the set case, use the method <code>set\_pwm\_duty\_cycle</code> () and provide the IO line configured as PWM and the value of the duty cycle in % of the PWM. The duty cycle is the proportion of 'ON' time to the regular interval or 'period' of time. A high duty cycle corresponds to high power, because the power is ON for most of the time. The percentage parameter of the set duty cycle method is a double, which allows you to be more precise in the configuration.

#### Set the duty cycle of an IO line configure as PWM

```
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

[...]

# Set a duty cycle of 75% to the DIO10_PWM0 line (PWM output).
local_xbee.set_pwm_duty_cycle(IOLine.DIO10_PWM0, 75)

[...]
```

The set\_pwm\_duty\_cycle () method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

The get\_pwm\_duty\_cycle (IOLine) method of a PWM line returns a double value with the current duty cycle percentage of the PWM.

#### Get the duty cycle of an IO line configured as PWM

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
(continues on next page)
```

(continues on next page)

```
# Get the duty cycle of the DIO10_PWM0 line (PWM output).
duty_cycle = local_xbee.get_pwm_duty_cycle(IOLine.DIO10_PWM0);
[...]
```

Note: In both cases (get and set), the IO line provided must be PWM capable and must be configured as PWM output.

# 2.6.7.2 Read IO samples

XBee modules can monitor and sample the analog and digital IO lines. You can read IO samples locally or transmitted to a remote device to provide an indication of the current IO line states.

There are three ways to obtain IO samples on a local or remote device:

- Queried sampling
- · Periodic sampling
- Change detection sampling

The XBee Python Library represents an IO sample by the IOSample class, which contains:

- Digital and analog channel masks that indicate which lines have sampling enabled.
- Values of those enabled lines.

You must configure the IO lines you want to receive in the IO samples before enabling sampling.

### **Queried sampling**

The XBee Python Library provides a method to read an IO sample that contains all enabled digital IO and analog input channels, read\_io\_sample(). The method returns an IOSample object.

### Read an IO sample and getting the DIO value

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
# Read an IO sample from the device.
io_sample = local_xbee.read_io_sample()
# Select the desired IO line.
io_line = IOLine.DIO3_AD3
# Check if the IO sample contains the expected IO line and value.
if io_sample.has_digital_value(io_line):
    print("DIO3 value: %s" % io_sample.get_digital_value(ioLine))
[...]
```

The read\_io\_sample() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

## Periodic sampling

Periodic sampling allows an XBee module to take an IO sample and transmit it to a remote device at a periodic rate. That remote device is defined in the destination address through the set\_dest\_address() method. The XBee Python Library provides the set\_io\_sampling\_rate() method to configure the periodic sampling.

The XBee module samples and transmits all enabled digital IO and analog inputs to the remote device every X seconds. A sample rate of 0 s disables this feature.

# Set the IO sampling rate

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
# Set the destination address.
local_xbee.set_dest_address(XBee64BitAddress.from_hex_string("0013A20040XXXXXX"))
# Set the IO sampling rate.
local_xbee.set_io_sampling_rate(5) # 5 seconds.
[...]
```

The set\_io\_sampling\_rate() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

You can also read this value using the <code>get\_io\_sampling\_rate()</code> method. This method returns the IO sampling rate in milliseconds and '0' when the feature is disabled.

### Get the IO sampling rate

```
[...]
# Instantiate an XBee device object.

(continues on next page)
```

```
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()

[...]

# Get the IO sampling rate.
value = local_xbee.get_io_sampling_rate()

[...]
```

The get\_io\_sampling\_rate() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

# 2.6.7.3 Change detection sampling

You can configure modules to transmit a data sample immediately whenever a monitored digital IO pin changes state. The set\_dio\_change\_detection() method establishes the set of digital IO lines that are monitored for change detection. A None set disables the change detection sampling.

As in the periodic sampling, change detection samples are transmitted to the configured destination address.

Note: This feature only monitors and samples digital IOs, so it is not valid for analog lines.

#### Set the DIO change detection

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
# Set the destination address.
local_xbee.set_dest_address(XBee64BitAddress.from_hex_string("0013A20040XXXXXX"))
# Create a set of IO lines to be monitored.
lines = [IOLine.DIO3_AD3, IOLine.DIO4_AD4]
# Enable the DIO change detection sampling.
local_xbee.set_dio_change_detection(lines)
[...]
```

The set\_dio\_change\_detection() method may fail for the following reasons:

• ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.

- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

You can also get the lines that are monitored using the get\_dio\_change\_detection() method. A None value indicates that this feature is disabled.

### Get the DIO change detection

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
# Get the set of lines that are monitored.
lines = local_xbee.get_dio_change_detection()
[...]
```

The get\_dio\_change\_detection() method may fail for the following reasons:

- ACK of the command sent is not received in the configured timeout, throwing a TimeoutException.
- Other errors caught as XBeeException:
  - The operating mode of the device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.
  - The response of the command is not valid, throwing an ATCommandException.
  - There is an error writing to the XBee interface, throwing a generic XBeeException.

## Register an IO sample listener

In addition to configuring an XBee device to monitor and sample the analog and digital IO lines, you must register a callback in the local device where you want to receive the IO samples. You are then notified when the device receives a new IO sample.

You must subscribe to the IO samples reception service by using the method add\_io\_sample\_received\_callback() with an IO sample reception callback function as parameter.

#### Add an IO sample callback

```
[...]
# Instantiate an XBee device object.
local_xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
# Define the IO sample receive callback.
```

(continues on next page)

```
def io_sample_callback(io_sample, remote_xbee, send_time):
    print("IO sample received at time %s." % str(send_time))
    print("IO sample:")
    print(str(io_sample))

# Subscribe to IO samples reception.
local_xbee.add_io_sample_received_callback(io_sample_callback)

[...]
```

This callback function will receive three parameters when an IO sample receive event is raised:

- The received IO sample as an IOSample object.
- The remote XBee device that sent the IO sample as a RemoteXBeeDevice object.
- The time in which the IO sample was received as an Float (calculated with Python standard time.time()).

To stop receiving notifications of new IO samples, remove the added callback using the del\_io\_sample\_received\_callback() method.

## Remove an IO sample callback

```
[...]
# Instantiate an XBee device object.
local xbee = XBeeDevice("COM1", 9600)
local_xbee.open()
[...]
# Define the IO sample receive callback.
def io_sample_callback(io_sample, remote_xbee, send_time):
   print("IO sample received at time %s." % str(send_time))
   print("IO sample:")
   print(str(io_sample))
# Subscribe to IO samples reception by adding the callback.
local_xbee.add_io_sample_received_callback(io_sample_callback)
[...]
# Unsubscribe from IO samples reception by removing the callback.
local_xbee.del_io_sample_received_callback(io_sample_callback)
[...]
```

The del\_io\_sample\_received\_callback () method will raise a ValueError if you try to delete a callback that you have not added yet.

# Example: Receive IO samples

The XBee Python Library includes a sample application that demonstrates how to configure a remote device to monitor IO lines and receive the IO samples in the local device. The example is located in the following path: **examples/io/IOSamplingSample/IOSamplingSample.py** 

# 2.6.8 Update the XBee

To keep your XBee devices up to date, the XBee Python Library provides several methods to update the device software including firmware, file system and XBee profiles:

- Update the XBee firmware
- Update the XBee file system
- Apply an XBee profile

### Warning:

At the moment, update features are only supported in:

- XBee 3:
  - Local and remote firmware updates
  - Local and remote file system updates
  - Local and remote profile updates
- XBee SX 868/900 MHz
  - Local and remote firmware updates
  - Local and remote profile updates
- XBee S2C
  - Remote firmware updates
  - Remote profile updates

# 2.6.8.1 Update the XBee firmware

You may need to update the running firmware of your XBee devices to, for example, change their XBee protocol, fix issues and security risks, or access to new features and functionality.

The XBee Python Library provides methods to perform firmware updates in local and remote devices:

- Update the firmware of a local XBee
- Update the firmware of a remote XBee

### Warning:

At the moment, firmware update is only supported in:

- XBee 3: Local and remote firmware updates
- XBee SX 868/900 MHz: Local and remote firmware updates
- XBee S2C: Remote firmware updates

### Update the firmware of a local XBee

The firmware update process of a local XBee device is performed over the serial connection. For this operation, you need the following components:

- The XBee device object instance or the serial port name where the device is attached to.
- The new firmware XML descriptor file.
- The new firmware binary file (\*.gbl)
- Optionally, the new bootloader binary file (\*.gbl) required by the new firmware.

Warning: Firmware update will fail if the firmware requires a new bootloader and it is not provided.

Warning: At the moment, local firmware update is only supported in XBee 3 and XBee SX 868/900 MHz devices.

## Example: Local Firmware Update

The XBee Python Library includes a sample application that displays how to perform a local firmware update. It can be located in the following path:

examples/firmware/LocalFirmwareUpdateSample/LocalFirmwareUpdateSample.py

# Update the local firmware using an XBee device object

If you have an object instance of your local XBee device, you have to call the update\_firmware method of the XBeeDevice class providing the required parameters:

Method		Description
update_firmware(String, String, String,	Integer,	Performs a firmware update operation of the device.
Function)	illegel,	<ul> <li>xml_firmware_file (String): path of the XML file that describes the firmware to upload.</li> <li>xbee_firmware_file (String, optional): location of the XBee binary firmware file (*.gbl).</li> <li>bootloader_firmware_file (String, optional): location of the bootloader binary firmware file (*.gbl).</li> <li>timeout (Integer, optional): the maximum amount of seconds to wait for target read operations during the update process.</li> <li>progress_callback (Function, optional): function to execute to receive progress information. Receives two arguments:         <ul> <li>The current update task as a String</li> <li>The current update task percentage as an Integer</li> </ul> </li> </ul>

The update\_firmware method may fail for the following reasons:

- The device does not support the firmware update operation, throwing a OperationNotSupportedException.
- There is an error during the firmware update operation, throwing a Firmware Update Exception.
- Other errors caught as XBeeException:

- The device is not open, throwing a generic XBeeException.
- The operating mode of the local XBee device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.

### Update local XBee device firmware using an XBee device object

# Update the local firmware using a serial port

If you do not know the XBee serial communication parameters or you cannot instantiate the XBee device object (for example if the device must be recovered), you can perform the firmware update process by providing the serial port identifier where the XBee is attached to.

In this scenario, use the update\_local\_firmware method of the XBee firmware module providing the required parameters. The library forces the XBee to reboot into bootloader mode, using the recovery mechanism, and performs the firmware update from that point.

Method	Description
update_local_firmware(String or XBeeDevice,	Performs a local firmware update operation in the given
String, String, Integer, Function)	target.
	<ul> <li>target (String or :class: '.XBeeDevice'): target of the firmware upload operation. * String: serial port identifier. * :class: '.AbstractXBeeDevice': the XBee device to upload its firmware.</li> <li>xml_firmware_file (String): path of the XML file that describes the firmware to upload.</li> <li>xbee_firmware_file (String, optional): location of the XBee binary firmware file (*.gbl).</li> <li>bootloader_firmware_file (String, optional): location of the bootloader binary firmware file.</li> <li>timeout (Integer, optional): the maximum amount of seconds to wait for target read operations during the update process.</li> <li>progress_callback (Function, optional): function to execute to receive progress information. Receives two arguments: <ul> <li>The current update task as a String</li> <li>The current update task percentage as an Integer</li> </ul> </li> </ul>

The update\_local\_firmware method may fail for the following reasons:

• There is an error during the firmware update operation, throwing a FirmwareUpdateException.

## Update local XBee device firmware using a serial port

94

# Update the firmware of a remote XBee

The firmware update process for remote XBee devices is performed over the air using special XBee frames. For this operation, you need the following components:

- The remote XBee device object instance.
- The new firmware XML descriptor file.
- The new firmware binary file (\*.ota)
- Optionally, the new firmware binary file with the bootloader embedded (\*.otb)

Warning: Firmware update fails if the firmware requires a new bootloader and the \*.otb file is not provided.

Warning: At the moment, remote firmware update is only supported in XBee 3, XBee SX 868/900 MHz, and XBee S2C devices.

To perform the remote firmware update, call the update\_firmware method of the RemoteXBeeDevice class providing the required parameters:

Method	Description
update_firmware(String, String, String, Integer, Function)	Performs a remote firmware update operation of the device.  • xml_firmware_file (String): path of the XML file that describes the firmware to upload.
	<ul> <li>xbee_firmware_file (String, optional): location of the XBee binary firmware file (*.ota).</li> <li>bootloader_firmware_file (String, optional): location of the XBee binary firmware file with bootloader embedded (*.otb).</li> <li>timeout (Integer, optional): the maximum amount of seconds to wait for target read operations during the update process.</li> <li>progress_callback (Function, optional): function to execute to receive progress information. Receives two arguments:         <ul> <li>The current update task as a String</li> <li>The current update task percentage as an Integer</li> </ul> </li> </ul>

The update\_firmware method may fail for the following reasons:

- The remote device does not support the firmware update operation, throwing a OperationNotSupportedException.
- There is an error during the firmware update operation, throwing a Firmware UpdateException.
- Other errors caught as XBeeException:
  - The local device is not open, throwing a generic XBeeException.
  - The operating mode of the local device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.

# Update remote XBee device firmware

```
[...]
XML_FIRMWARE_FILE = "my_path/my_firmware.xml"
OTA_FIRMWARE_FILE = "my_path/my_firmware.ota"
OTB_FIRMWARE_FILE = "my_path/my_firmware.otb"
REMOTE_DEVICE_NAME = "REMOTE"
[...]
# Instantiate an XBee device object.
xbee = XBeeDevice(...)
# Get the network.
xnet = xbee.get_network()
# Get the remote device.
remote = xnet.discover_device(REMOTE_DEVICE_NAME)
# Update the remote XBee device firmware.
remote.update_firmware(SERIAL_PORT,
                       XML_FIRMWARE_FILE,
                       xbee_firmware_file=OTA_FIRMWARE_FILE,
                       bootloader_firmware_file=OTB_FIRMWARE_FILE,
                       progress_callback=progress_callback,)
[...]
```

# Example: Remote Firmware Update

The XBee Python Library includes a sample application that displays how to perform a remote firmware update. It can be located in the following path:

examples/firmware/RemoteFirmwareUpdateSample/RemoteFirmwareUpdateSample.py

# 2.6.8.2 Update the XBee file system

XBee 3 devices feature file system capabilities, meaning that they are able to persistently store files and folders in flash. The XBee Python Library provides classes and methods to manage these files.

- Create file system manager
- File system operations

Warning: At the moment file system capabilities are only supported in XBee 3 devices.

#### Create file system manager

A LocalXBeeFileSystemManager object is required to work with local devices file system. You can instantiate this class by providing the local XBee device object. Once you have the object instance, you must call the connect method to open the file system connection and leave it ready to work.

**Warning:** File system operations take ownership of the serial port, meaning that you will stop receiving messages from the device until file system connection is closed. For this reason it is highly recommended to call the disconnect method of the file system manager as soon as you finish working with it.

Method	Description
connect()	Connects the file system manager.
disconnect()	Disconnects the file system manager and restores the device connection.

The connect method may fail for the following reasons:

- The device does not support the file system capabilities, throwing a FileSystemNotSupportedException.
- There is an error during the connect operation, throwing a FileSystemException.

# Create a local file system manager

```
from digi.xbee.filesystem import LocalXBeeFileSystemManager
[...]
# Instantiate an XBee device object.
xbee = XBeeDevice(...)
[...]
# Create the file system manager and connect it.
filesystem_manager = LocalXBeeFileSystemManager(xbee)
filesystem_manager.connect()
[...]
filesystem_manager.disconnect()
```

### File system operations

The file system manager provides several methods to navigate through the device file system and operate with the different files and folders:

Method	Description
get_current_directory()	Returns the current device directory.
change_directory(String)	Changes the current device working directory to the given one.  • directory (String): the new directory to change to.
make_directory(String)	Creates the provided directory.  • directory (String): the new directory to create.
list_directory(String)	Lists the contents of the given directory.  • directory (String, optional): the directory to list its contents. Optional. If not provided, the current directory contents are listed.
remove_element(String)	Removes the given file system element path.  • element_path (String): path of the file system element to remove.
move_element(String, String)	Moves the given source element to the given destination path.  • source_path (String): source path of the element to move.  • dest_path (String): destination path of the element to move.
put_file(String, String, Boolean, Function)	Transfers the given file in the specified destination path of the XBee device.  • source_path (String): the path of the file to transfer.  • dest_path (String): the destination path to put the file in.  • secure (Boolean, optional): True if the file should be stored securely, False otherwise. Defaults to False.  • progress_callback (Function, optional): function to execute to receive progress information. Takes the following arguments:  - The progress percentage as integer.
put_dir(String, String, Function)	Uploads the given source directory contents into the given destination directory in the device.  • source_dir (String): the local directory to upload its contents.  • dest_dir (String, optional): the remote directory to upload the contents to. Defaults to current directory.  • progress_callback (Function, optional): function to execute to receive progress information. Takes the following arguments:  - The file being uploaded as string.  - The progress percentage as integer.
get_file(String, String, Function)	Downloads the given XBee device file in the specified destination path.
98	<ul> <li>source_path (String): the hapterthe contents vice file to download.</li> <li>dest_path (String): the destination path to store the file in</li> </ul>

the file in.

The methods above may fail for the following reasons:

• There is an error executing the requested operation, throwing a FileSystemException.

#### Example: Format file system

The XBee Python Library includes a sample application that displays how to format the device file system. It can be located in the following path:

examples/filesystem/FormatFilesystemSample/FormatFilesystemSample.py

#### Example: List directory

The XBee Python Library includes a sample application that displays how to list the contents of a device directory. It can be located in the following path:

examples/filesystem/ListDirectorySample/ListDirectorySample.py

### Example: Upload/download file

The XBee Python Library includes a sample application that displays how to upload/download a file from the device. It can be located in the following path:

examples/filesystem/UploadDownloadFileSample/UploadDownloadFileSample.py

# 2.6.8.3 Apply an XBee profile

An XBee profile is a snapshot of a specific XBee configuration, including firmware, settings, and file system contents. The XBee Python API includes a set of classes and methods to work with XBee profiles and apply them to local and remote devices.

- Read an XBee profile
- Apply an XBee profile to a local device
- Apply an XBee profile to a remote device

To configure individual settings see Configure the XBee device.

Note: Use XCTU to create configuration profiles.

# Warning:

At the moment, firmware update is only supported in:

- **XBee 3**: Local and remote profile updates
- XBee SX 868/900 MHz: Local and remote profile updates
- XBee S2C: Remote profile updates

#### Read an XBee profile

The library provides a class called XBeeProfile that is used to read and extract information of an existing XBee profile file.

To create an XBeeProfile object, provide the location of the profile file in the class constructor.

### Instantiate a profile

```
from digi.xbee.profile import XBeeProfile

[...]

PROFILE_PATH = "/home/user/my_profile.xpro"

[...]

# Create the XBee profile object.
xbee_profile = XBeeProfile(PROFILE_PATH)

[...]
```

The creation of the XBee profile object may fail for the following reasons:

- The provided profile file is not valid, throwing a ValueError.
- There is any error reading the profile file, throwing a ProfileReadException.

Once the XBee profile object is created, you can extract some profile information by accessing each of the exposed properties:

Property	Description
profile_file	Returns the profile file.
version	Returns the profile version.
flash_firmware_option	Returns the profile flash firmware option.
description	Returns the profile description.
reset_settings	Returns whether the settings of the XBee device are reset before applying the profile
	ones.
has_firmware_files	Returns whether the profile has firmware binaries (local or remote)
has_local_firmware_files	Returns whether the profile has local firmware binaries.
has_remote_firmware_files	Returns whether the profile has remote firmware binaries.
has_filesystem	Returns whether the profile has filesystem information (local or remote)
has_local_filesystem	Returns whether the profile has local filesystem information.
has_remote_filesystem	Returns whether the profile has remote filesystem information.
profile_settings	Returns all the firmware settings that the profile configures.
firmware_version	Returns the compatible firmware version of the profile.
hardware_version	Returns the compatible hardware version of the profile.
compatibility_number	Returns the compatibility number of the profile.
region_lock	Returns the region lock of the profile.

To access to the files inside, use open method. Once done with it, use close method.

### Open/close a profile

```
xbee_profile = XBeeProfile(PROFILE_PATH)
xbee_profile.open()
[...]
xbee_profile.close()
[...]
```

An opened profile also offers the following properties:

Property	Description	
profile_description_file	Returns the path of the profile description file.	
firmware_description_file	Returns the path of the profile firmware description file.	
file_system_path	Returns the profile file system path.	
remote_file_system_image	Returns the path of the remote OTA file system image.	
bootloader_file	Returns the profile bootloader file path.	

#### Read a profile

```
from digi.xbee.profile import XBeeProfile

[...]

PROFILE_PATH = "/home/user/my_profile.xpro"

[...]

# Create the XBee profile object.
xbee_profile = XBeeProfile(PROFILE_PATH)

# Print profile compatible hardware and software versions
print(" - Firmware version: %s" % xbee_profile.firmware_version)
print(" - Hardware version: %s" % xbee_profile.hardware_version)

[...]
```

Example: Read an XBee profile

The XBee Python Library includes a sample application that displays how to read an XBee profile. It can be located in the following path:

examples/profile/ReadXBeeProfileSample/ReadXBeeProfileSample.py

# Apply an XBee profile to a local device

Applying a profile to a local XBee device requires the following components:

- The local XBee device object instance.
- The profile file to apply (\*.xpro).

**Note:** Use XCTU to create configuration profiles.

Warning: At the moment, local profile update is only supported in XBee 3 and XBee SX 868/900 MHz devices.

To apply the XBee profile to a local XBee, you have to call the apply\_profile method of the XBeeDevice class providing the required parameters:

Method	Description
apply_profile(String, timeout, Function)	Applies the given XBee profile to the XBee device.
	• profile_path (String): path of the XBee profile
	file to apply.
	• timeout (Integer, optional): maximum time to
	wait for read operations during the apply profile.
	• progress_callback (Function, optional): func-
	tion to execute to receive progress information.
	Receives two arguments:
	<ul> <li>The current apply profile task as a String</li> </ul>
	<ul> <li>The current apply profile task percentage as</li> </ul>
	an Integer
	_

The apply\_profile method may fail for the following reasons:

- The local device does not support the apply profile operation, throwing a OperationNotSupportedException.
- There is an error while applying the XBee profile, throwing a UpdateProfileException.
- Other errors caught as XBeeException:
  - The local device is not open, throwing a generic XBeeException.
  - The operating mode of the local device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.

# Apply an XBee profile to a local device

```
[...]

PROFILE_PATH = "/home/user/my_profile.xpro"

[...]

# Instantiate an XBee device object.
xbee = XBeeDevice(...)

[...]

# Apply the XBee device profile.
device.apply_profile(PROFILE_PATH, progress_callback=progress_callback)

[...]
```

# Example: Apply local XBee profile

The XBee Python Library includes a sample application that displays how to apply an XBee profile to a local device. It can be located in the following path:

examples/profile/ApplyXBeeProfileSample/ApplyXBeeProfileSample.py

# Apply an XBee profile to a remote device

Applying a profile to a remote XBee requires the following components:

• The remote XBee device object instance.

102

• The profile file to apply (\*.xpro).

**Note:** Use XCTU to create configuration profiles.

Warning: At the moment, remote profile update is only supported in XBee 3, XBee SX 868/900 MHz, and XBee S2C devices.

To apply the XBee profile to a remote XBee device, you have to call the apply\_profile method of the RemoteXBeeDevice class providing the required parameters:

Method	Description
apply_profile(String, timeout, Function)	Applies the given XBee profile to the remote XBee de-
	vice.
	• profile_path (String): path of the XBee profile
	file to apply.
	• timeout (Integer, optional): maximum time to
	wait for read operations during the apply profile.
	• progress_callback (Function, optional): func-
	tion to execute to receive progress information.
	Receives two arguments:
	<ul> <li>The current apply profile task as a String</li> </ul>
	<ul> <li>The current apply profile task percentage as</li> </ul>
	an Integer

The apply\_profile method may fail for the following reasons:

- The remote device does not support the apply profile operation, throwing a OperationNotSupportedException.
- There is an error while applying the XBee profile, throwing a UpdateProfileException.
- Other errors caught as XBeeException:
  - The local device is not open, throwing a generic XBeeException.
  - The operating mode of the local device is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.

## Apply an XBee profile to a remote device

```
[...]
PROFILE_PATH = "/home/user/my_profile.xpro"
REMOTE_DEVICE_NAME = "REMOTE"

[...]
# Instantiate an XBee device object.
xbee = XBeeDevice(...)
# Get the network.
xnet = xbee.get_network()
```

(continues on next page)

(continued from previous page)

```
# Get the remote device.
remote = xnet.discover_device(REMOTE_DEVICE_NAME)

[...]

# Apply the XBee device profile.
remote.apply_profile(PROFILE_PATH, progress_callback=progress_callback)

[...]
```

### Example: Apply remote XBee profile

The XBee Python Library includes a sample application that displays how to apply an XBee profile to a remote device. It can be located in the following path:

examples/profile/ApplyXBeeProfileRemoteSample/ApplyXBeeProfileRemoteSample.py

# 2.6.9 Log events

Logging is a fundamental part of applications, and every application includes this feature. A well-designed logging system is a useful utility for system administrators, developers, and the support team and can save valuable time in sorting through the cause of issues. As users execute programs on the front end, the system invisibly builds a vault of event information (log entries).

The XBee Python Library uses the Python standard logging module for registering logging events. The logger works at module level; that is, each module has a logger with a unique name.

The modules that have logging integrated are devices and reader. By default, all loggers are disabled so you will not see any logging message in the console if you do not activate them.

In the XBee Python Library, you need three things to enable the logger:

- 1. The logger itself.
- 2. A handler. This will determine if the messages will be displayed in the console, written in a file, sent through a socket, etc.
- 3. A formatter. This will determine the message format. For example, a format could be:
  - Timestamp with the current date logger name level (debug, info, warning...) data.

To retrieve the logger, use the <code>get\_logger()</code> method of the logging module, providing the name of the logger that you want to get as parameter. In the XBee Python Library all loggers have the name of the module they belong to. For example, the name of the logger of the <code>devices</code> module is <code>digi.xbee.devices</code>. You can get a module name with the special attribute <code>\\_\_name\\_\_</code>.

# Retrieve a module name and its logger

```
import logging
[...]

# Get the logger of the devices module.
dev_logger = logging.getLogger(digi.xbee.devices.__name__)

# Get the logger of the devices module providing the name.
dev_logger = logging.getLogger("digi.xbee.devices")
[...]
```

To retrieve a handler, you can use the default Python handler or create your own one. Depending on which type of handler you use, the messages created by the logger will be printed in the console, in a file, etc. You can have more than one handler per logger, this means that you can enable the default XBee Python Library handler and add your own handlers.

## Retrieve a handler and add it to a logger

```
import logging
[...]

# Get the logger of the devices module.
dev_logger = logging.getLogger(digi.xbee.devices.__name__)

# Get a handler and add it to the logger.
handler = logging.StreamHandler()
dev_logger.addHandler(handler)

[...]
```

The previous code snippet shows how to add a handler to a logger, but the logical way is to add a formatter to a handler, and then add the handler to the logger.

When you create a formatter, you must specify which information will be printed and in which format. This guide shows you how to create a formatter with a simple format. If you want to create more complex formatters or handlers, see the Python documentation.

### Create a formatter and add it to a handler

# Enable a logger for the devices module

(continues on next page)

(continued from previous page)

```
handler.setFormatter(formatter)

# Add the handler to the logger.
dev_logger.addHandler(handler)

[...]
```

# 2.6.9.1 Logging level

The XBee Python Library also provides a method in the utils module, enable\_logger(), to enable the logger with the default settings. These settings are:

- Handler: StreamHandler
- Format: timestamp logger name level message

Method	Description
enable_logger(name, level=logging.DEBUG)	Enables the logger.
	<ul> <li>name: the name of the module whose logger you want to activate.</li> <li>level: default DEBUG. The level you want to see.</li> </ul>

# Enable a logger

```
import logging
[...]

# Enable the logger in the digi.xbee.devices module with INFO level.
dev_logger = enable_logger(digi.xbee.devices.__name__, logging.INFO)

# This is a valid method to do the same.
dev_logger = enable_logger("digi.xbee.devices", logging.INFO)

[...]

# Enable the logger in the digi.xbee.devices module with the default level
# (DEBUG).
dev_logger = enable_logger("digi.xbee.devices")

# This is a valid method to do the same.
dev_logger = enable_logger("digi.xbee.devices", logging.DEBUG)

[...]
```

**Note:** For further information about the Python logging module, see the Python logging module official documentation or the Python logging cookbook.

# 2.6.10 XBee Python samples

The XBee Python Library includes several samples to demonstrate how to do the following:

- Communicate with your modules
- · Configure your modules
- · Read the IO lines
- Update device's firmware
- Work with device's file system
- Apply XBee profiles
- Perform other common operations

All of the sample applications are contained in the examples folder, organized by category. Every sample includes the source code and a **readme.txt** file to clarify the purpose and the required setup to launch the application.

Examples are split by categories:

- Configuration samples
- · Network samples
- Communication samples
- IO samples
- Firmware samples
- File system samples
- Profile samples

## 2.6.10.1 Configuration samples

# Manage common parameters

This sample application shows how to get and set common parameters of the XBee device. Common parameters are split in cached and non-cached parameters. For that reason, the application refreshes the cached parameters before reading and displaying them. The application then configures, reads, and displays the value of non-cached parameters.

The application uses the specific setters and getters provided by the XBee device object to configure and read the different parameters.

You can locate the example in the following path: examples/configuration/ManageCommonParametersSample

**Note:** For more information about how to manage common parameters, see *Read and set common parameters*.

# Set and get parameters

This sample application shows how to set and get parameters of a local or remote XBee device. Use this method when you need to set or get the value of a parameter that does not have its own getter and setter within the XBee device object.

The application sets the value of four parameters with different value types:

- String
- Byte
- Array

Integer

The application then reads the parameters from the device to verify that the read values are the same as the values that were set.

You can locate the example in the following path: examples/configuration/SetAndGetParametersSample

**Note:** For more information about how to get and set other parameters, see *Read, set and execute other parameters*.

#### Reset module

This sample application shows how to perform a software reset on the local XBee module.

You can locate the example in the following path: examples/configuration/ResetModuleSample

**Note:** For more information about how to reset a module, see *Reset the device*.

### **Recover XBee serial connection**

This sample application shows how to recover the serial settings of a local XBee.

You can locate the example at the following path: examples/configuration/RecoverSerialConnection

**Note:** For more information about this, see *Open the XBee device connection*.

# Connect to access point (Wi-Fi)

This sample application shows how to configure a Wi-Fi module to connect to a specific access point and read its addressing settings.

You can locate the example at the following path: examples/configuration/ConnectToAccessPoint

**Note:** For more information about connecting to an access point, see *Configure Wi-Fi settings*.

# 2.6.10.2 Network samples

### **Discover devices**

This sample application demonstrates how to obtain the XBee network object from a local XBee device and discover the remote XBee devices that compose the network. The example adds a discovery listener, so the callbacks provided by the listener object receive the events.

The remote XBee devices are printed out as soon as they are found during discovery.

You can locate the example in the following path: examples/network/DiscoverDevicesSample

**Note:** For more information about how to perform a network discovery, see *Discover the network*.

# **Network modifications sample**

This sample application demonstrates how to listen to network modification events. The example adds a modifications network callback, so modifications events are received and printed out.

A network is modified when:

- a new node is added by discovering, manually, or because data is received from it
- an existing node is removed from the network
- · an existing node is updated with new information
- · it is fully cleared

You can locate the example in the following path: examples/network/NetworkModificationsSample

**Note:** For more information about how to listen to network modifications, see *Listen to network modification events*.

# 2.6.10.3 Communication samples

## Send data

This sample application shows how to send data from the XBee device to another remote device on the same network using the XBee Python Library. In this example, the application sends data using a reliable transmission method. The application blocks during the transmission request, but you are notified if there is any error during the process.

The application sends data to a remote XBee device on the network with a specific node identifier (name).

You can locate the example in the following path: examples/communication/SendDataSample

**Note:** For more information about how to send data, see *Send data*.

### Send data asynchronously

This sample application shows how to send data asynchronously from the XBee device to another remote device on the same network using the XBee Python Library. Transmitting data asynchronously means the execution is not blocked during the transmit request, but you cannot determine if the data was successfully sent.

The application sends data asynchronously to a remote XBee device on the network with a specific node identifier (name).

You can locate the example in the following path: examples/communication/SendDataAsyncSample

**Note:** For more information about how to send data, see *Send data*.

### Send broadcast data

This sample application shows how to send data from the local XBee device to all remote devices on the same network (broadcast) using the XBee Python Library. The application blocks during the transmission request, but you are notified if there is any error during the process.

You can locate the example in the following path: examples/communication/SendBroadcastDataSample

**Note:** For more information about how to send broadcast data, see *Send data to all devices of the network*.

# Send explicit data

This sample application shows how to send data in the application layer (explicit) format to a remote Zigbee device using the XBee Python Library. In this example, the XBee module sends explicit data using a reliable transmission method. The application blocks during the transmission request, but you are notified if there is any error during the process.

You can locate the example in the following path: examples/communication/explicit/SendExplicitDataSample

**Note:** For more information about how to send explicit data, see *Send explicit data*.

# Send explicit data asynchronously

This sample application shows how to send data in the application layer (explicit) format asynchronously to a remote Zigbee device using the XBee Python Library. Transmitting data asynchronously means the execution is not blocked during the transmit request, but you cannot determine if the data was successfully sent.

You can locate the example in the following path: examples/communication/explicit/SendExplicitDataAsyncSample

**Note:** For more information about how to send explicit data, see *Send explicit data*.

### Send broadcast explicit data

This sample application shows how to send data in the application layer (explicit) format to all remote devices on the network (broadcast) using the XBee Python Library. The application blocks during the transmission request, but you are notified if there is any error during the process.

You can locate the example in the following path: examples/communication/explicit/SendBroadcastExplicitDataSample

**Note:** For more information about how to send broadcast explicit data, see *Send explicit data to all devices in the network*.

# Send IP data (IP devices)

This sample application shows how to send IP data to another device specified by its IP address and port number.

You can find the example at the following path: examples/communication/ip/SendIPDataSample

**Note:** For more information about how to send IP data, see *Send IP data*.

### Send SMS (cellular devices)

This sample application shows how to send an SMS to a phone or cellular device.

You can find the example at the following path: examples/communication/cellular/SendSMSSample

**Note:** For more information about how to send SMS messages, see *Send SMS messages*.

# Send UDP data (IP devices)

This sample application shows how to send UDP data to another device specified by its IP address and port number.

You can find the example at the following path: examples/communication/ip/SendUDPDataSample

**Note:** For more information about how to send IP data, see *Send IP data*.

# **Send Bluetooth Data**

This sample application shows how to send data to the XBee Bluetooth Low Energy interface.

You can find the example at the following path: examples/communication/bluetooth/SendBluetoothDataSample

Note: For more information about sending Bluetooth data, see Send Bluetooth data.

# **Send MicroPython Data**

This sample application shows how to send data to the XBee MicroPython interface.

You can find the example at the following path: examples/communication/micropython/SendMicroPythonDataSample

Note: For more information about sending MicroPython data, see Send MicroPython data.

## Send User Data Relay

This sample application shows how to send data to other XBee interface.

You can find the example at the following path: examples/communication/relay/SendUserDataRelaySample

**Note:** For more information about sending User Data Relay messages, see *Send Bluetooth data* or *Send MicroPython data*.

#### Receive data

This sample application shows how data packets are received from another XBee device on the same network.

The application prints the received data to the standard output in ASCII and hexadecimal formats after the sender address.

You can locate the example in the following path: examples/communication/ReceiveDataSample

**Note:** For more information about how to receive data using a callback, see *Data reception callback*.

# Receive data polling

This sample application shows how data packets are received from another XBee device on the same network using a polling mechanism.

The application prints the data that was received to the standard output in ASCII and hexadecimal formats after the sender address.

You can locate the example in the following path: examples/communication/ReceiveDataPollingSample

**Note:** For more information about how to receive data using a polling mechanism, see *Polling for data*.

# Receive explicit data

This sample application shows how a Zigbee device receives data in the application layer (explicit) format using a callback executed every time new data is received. Before receiving data in explicit format, the API output mode of the Zigbee device is configured in explicit mode.

You can locate the example in the following path: examples/communication/explicit/ReceiveExplicitDataSample

**Note:** For more information about how to receive explicit data using a callback, see *Explicit data reception callback*.

## Receive explicit data polling

This sample application shows how a Zigbee device receives data in the application layer (explicit) format using a polling mechanism. Before receiving data in explicit format, the API output mode of the Zigbee device is configured in explicit mode.

You can locate the example in the following path: examples/communication/explicit/ReceiveExplicitDataPollingSample

**Note:** For more information about how to receive explicit data using a polling mechanism, see *Polling for explicit data*.

# Receive IP data (IP devices)

This sample application shows how an IP device receives IP data using a callback executed every time it receives new IP data.

You can find the example at the following path: examples/communication/ip/ReceiveIPDataSample

Note: For more information about how to receive IP data using a polling mechanism, see Receive IP data.

# Receive SMS (cellular devices)

This sample application shows how to receive SMS messages configuring a callback executed when new SMS is received.

You can find the example at the following path: examples/communication/cellular/ReceiveSMSSample

**Note:** For more information about how to receive SMS messages, see *Receive SMS messages*.

#### Receive Bluetooth data

This sample application shows how to receive data from the XBee Bluetooth Low Energy interface.

You can find the example at the following path: examples/communication/bluetooth/ReceiveBluetoothDataSample

Note: For more information about receiving Bluetooth data, see Receive Bluetooth data.

# Receive Bluetooth file

This sample application shows how to receive a file from the XBee Bluetooth Low Energy interface.

You can find the example at the following path: examples/communication/bluetooth/ReceiveBluetoothFileSample

Note: For more information about receiving Bluetooth data, see Receive Bluetooth data.

# **Receive MicroPython data**

This sample application shows how to receive data from the XBee MicroPython interface.

You can find the example at the following path: examples/communication/micropython/ReceiveMicroPythonDataSample

Note: For more information about receiving MicroPython data, see Receive MicroPython data.

# **Receive User Data Relay**

This sample application shows how to receive data from other XBee interface.

You can find the example at the following path: examples/communication/relay/ReceiveUserDataRelaySample

**Note:** For more information about receiving User Data Relay messages, see *Receive Bluetooth data* or *Receive MicroPython data*.

#### Receive modem status

This sample application shows how modem status packets (events related to the device and the network) are handled using the API.

The application prints the modem status events to the standard output when received.

You can locate the example in the following path: examples/communication/ReceiveModemStatusSample

**Note:** For more information about how to receive modem status events, see *Receive modem status events*.

### Connect to echo server (IP devices)

This sample application shows how IP devices can connect to an echo server, send data to it and reads the echoed data.

You can find the example at the following path: examples/communication/ip/ConnectToEchoServerSample

Note: For more information about how to send and receive IP data, see Send IP data and Receive IP data.

## Create a TCP client socket (cellular devices)

This sample application shows how to create a TCP client socket to send HTTP requests.

You can find the example at the following path: examples/communication/socket/SocketTCPClientSample

**Note:** For more information about how to use the XBee socket API, see *Communicate using XBee sockets*.

# Create a TCP server socket (cellular devices)

This sample application shows how to create a TCP server socket to receive data from incoming sockets.

You can find the example at the following path: examples/communication/socket/SocketTCPServerSample

**Note:** For more information about how to use the XBee socket API, see *Communicate using XBee sockets*.

# Create a UDP server/client socket (cellular devices)

This sample application shows how to create a UDP socket to deliver messages to a server and listen for data coming from multiple peers.

You can find the example at the following path: examples/communication/socket/SocketUDPServerClientSample

Note: For more information about how to use the XBee socket API, see Communicate using XBee sockets.

# 2.6.10.4 IO samples

### **Local DIO**

This sample application shows how to set and read XBee digital lines of the device attached to the serial/USB port of your PC.

The application configures two IO lines of the XBee device: one as a digital input (button) and the other as a digital output (LED). The application reads the status of the input line periodically and updates the output to follow the input.

The LED lights up while you press the button.

You can locate the example in the following path: examples/io/LocalDIOSample

Note: For more information about how to set and read digital lines, see Digital Input/Output.

### **Local ADC**

This sample application shows how to read XBee analog inputs of the device attached to the serial/USB port of your PC.

The application configures an IO line of the XBee device as ADC. It periodically reads its value and prints it in the output console.

You can locate the example in the following path: examples/io/LocalADCSample

**Note:** For more information about how to read analog lines, see *ADC*.

## **Remote DIO**

This sample application shows how to set and read XBee digital lines of remote devices.

The application configures two IO lines of the XBee devices: one in the remote device as a digital input (button) and the other in the local device as a digital output (LED). The application reads the status of the input line periodically and updates the output to follow the input.

The LED lights up while you press the button.

You can locate the example in the following path: examples/io/RemoteDIOSample

Note: For more information about how to set and read digital lines, see Digital Input/Output.

## **Remote ADC**

This sample application shows how to read XBee analog inputs of remote XBee devices.

The application configures an IO line of the remote XBee device as ADC. It periodically reads its value and prints it in the output console.

You can locate the example in the following path: examples/io/RemoteADCSample

**Note:** For more information about how to read analog lines, see *ADC*.

# IO sampling

This sample application shows how to configure a remote device to send automatic IO samples and how to read them from the local module.

The application configures two IO lines of the remote XBee device: one as digital input (button) and the other as ADC, and enables periodic sampling and change detection. The device sends a sample every five seconds containing the values of the two monitored lines. The device sends another sample every time the button is pressed or released, which only contains the value of this digital line.

The application registers a listener in the local device to receive and handle all IO samples sent by the remote XBee module.

You can locate the example in the following path: examples/io/IOSamplingSample

**Note:** For more information about how to read IO samples, see *Read IO samples*.

## 2.6.10.5 Firmware samples

## **Update local firmware**

This sample Python application shows how to update the firmware of a local XBee device.

The application provides the required hardware files to the update method as well as a callback function to be notified of progress.

You can locate the example in the following path: examples/firmware/LocalFirmwareUpdateSample

# Update remote firmware

This sample Python application shows how to update the firmware of a remote XBee device.

The application provides the required hardware files to the update method as well as a callback function to be notified of progress.

You can locate the example in the following path: examples/firmware/RemotelFirmwareUpdateSample

# 2.6.10.6 File system samples

# Format file system

This sample Python application shows how to format the filesystem of a local XBee device and retrieve usage information.

The application uses the LocalXBeeFileSystemManager to access the device filesystem and execute the required actions.

You can locate the example in the following path: examples/filesystem/FormatFilesystemSample

# List directory contents

This sample Python application shows how to list the contents of an XBee device filesystem directory.

The application uses the LocalXBeeFileSystemManager to access the device filesystem and executes the required actions.

You can locate the example in the following path: examples/filesystem/ListDirectorySample

## Upload/download file

This sample Python application shows how to upload and download a file from a local XBee device filesystem.

The application uses the LocalXBeeFileSystemManager to access the device filesystem and provides the local file and the necessary paths to the upload/download methods as well as callback functions to be notified of progress.

You can locate the example in the following path: examples/filesystem/UploadDownloadFileSample

# 2.6.10.7 Profile samples

### Apply local profile

This sample Python application shows how to apply an existing XBee profile to a XBee device.

The application provides the profile file to the update method as well as a callback function to be notified of progress.

You can locate the example in the following path: examples/profile/ApplyXBeeProfileSample

### Apply remote profile

This sample Python application shows how to apply an existing XBee profile to a remote XBee device.

The application provides the profile file to the update method as well as a callback function to be notified of progress.

You can locate the example in the following path: examples/profile/ApplyXBeeProfileRemoteSample

# Read profile

This sample Python application shows how to read an existing XBee profile and extract its properties.

The application creates an XBee profile object from an existing XBee profile file and prints all the accessible settings and properties.

You can locate the example in the following path: examples/profile/ReadXBeeProfileSample

# 2.6.11 Frequently Asked Questions (FAQs)

The FAQ section contains answers to general questions related to the XBee Python Library.

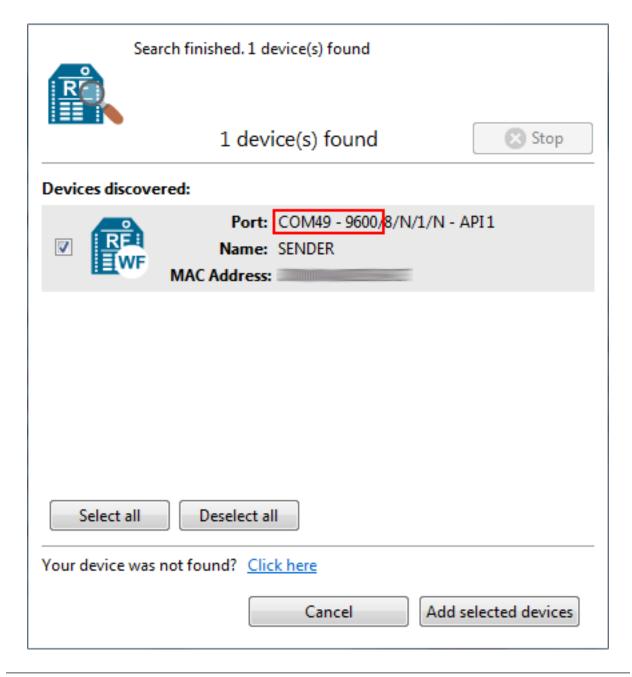
## 2.6.11.1 What is XCTU and how do I download it?

XCTU is a free multi-platform application designed to enable developers to interact with Digi RF modules through a simple-to-use graphical interface. You can download it at www.digi.com/xctu.

# 2.6.11.2 How do I find the serial port and baud rate of my module?

Open the XCTU application, and click the **Discover radio modules connected to your machine** button.

Select all ports to be scanned, click **Next** and then **Finish**. Once the discovery process has finished, a new window notifies you how many devices have been found and their details. The serial port and the baud rate are shown in the **Port** label.



**Note:** Note In UNIX systems, the complete name of the serial port contains the /dev/ prefix.

# 2.6.11.3 Can I use the XBee Python Library with modules in AT operating mode?

No, the XBee Python Library only supports **API** and **API Escaped** operating modes.

# 2.6.11.4 I get the Python error ImportError: No module named 'serial'

This error means that Python cannot find the serial module, which is used by the library for the serial communication with the XBee devices.

You can install PySerial running this command in your terminal application:

```
$ pip install pyserial
```

For further information about the installation of PySerial, refer to the PySerial installation guide.

# 2.6.11.5 | get the Python error ImportError: No module named 'srp'

This error means that Python cannot find the srp module, which is used by the library to authenticate with XBee devices over Bluetooth Low Energy.

You can install SRP running this command in your terminal application:

```
$ pip install srp
```

# 2.6.12 Changelog

### 2.6.12.1 v1.4.0 - 03/18/2021

- Deep node discovery for Zigbee, DigiMesh, and 802.15.4.
- Get route from local XBee to a remote XBee:
  - New method to register a callback to listen for new received routes (add\_route\_received\_callback())
  - New blocking method to ask for the route to the remote node (get\_route\_to\_node())
- Allow to recover a local node from a profile not only from firmware.
- Support to be notified when new frames are received from a specific node (add\_packet\_received\_from\_callback()).
- Update network information from sent/received AT Command frames.
- New optional argument for parameter value in execute\_command().
- New optional argument to apply pending settings in get\_parameter(), set\_parameter(), and execute\_command().
- XBee 3:
  - Support to update remote file system OTA images.
- XBee SX 900/868:
  - Firmware update for local and remote XBee devices.
  - Profile update for local and remote XBee devices.
- XBee S2C:
  - OTA firmware/profile update support for remote nodes.
- Zigbee:
  - Methods to get nodes routing and neighbor tables: get\_routes() and get\_neighbors().
  - Methods to get/set many-to-one broadcasting time: get\_many\_to\_one\_broadcasting\_time()
     and set\_many\_to\_one\_broadcasting\_time().
  - Support for source route creation: create\_source\_route().

- New frames: \* 'Route Record Indicator' (0xA1) \* 'Create Source Route Packet' (0x21)
- DigiMesh:
  - Method to get node neighbors: get\_neighbors().
  - Method to build aggregate route: build\_aggregate\_routes().
  - New frames: \* 'Route Information Packet' (0x8D)
- Documentation update
- Bug fixing:
  - Captured possible exception while determining the XBee role (#103)
  - Memory leak: empty list of last discovered nodes using ND (#172)
  - Fix Python 3.9 syntax error (#204)
  - Use least significant nibble of status field in local/remote AT Command Responses (XCTUNG-376)
  - Do not lose already registered socket callbacks when closing a local XBee.
  - Reload node information after firmware/profile update (XBPL-348)
  - OTA firmware update:
    - \* Fix sequence number in ZCL responses during fw update (XCTUNG-1975)
    - \* Immediate update after transferring the OTA file (XBPL-350)
    - \* Use requested file offset and size instead of fixed chunks (XBPL-344)
    - \* Mechanism to calculate the proper block size based on the maximum size received by the client and the maximum payload size (XBPL-346)
    - \* For asyncronous sleeping nodes (Zigbee, DigiMesh, 802.15.4) and synchronous sleeping networks (DigiMesh), configure a minimum sleep time before update and restore settings at the end. For DigiMesh synchronous sleeping network, the local XBee must be a non-sleeping node but synchronized with the network (SM=7)
  - Profile application:
    - \* Do not uncompress profile when reading its information. This change avoids extra processing time and required space when retrieving profile info.
    - \* Remove profile extracted files. A profile is opened to access to its contents, and must be closed when done with it.
    - \* Fixed the application of XBee profiles with 'AP' setting changes (XBPL-340)
    - \* Fixed bootloader update from profile due to bootloader image path mismatch (XBPL-338)
    - \* Fix bootloader update operation by waiting some time until the new bootloader is running (XBPL-339)
    - \* Fixed application of profile with filesystem from Windows(XBPL-341)
    - \* Read firmware version as an hexadecimal value (#177)
  - Several minor bug fixes.

## 2.6.12.2 v1.3.0 - 11/05/2019

- Zigbee: Support to register joining devices to a trust center.
- Cellular: XBee TCP/UDP socket support.
- XBee 3:
  - Firmware update for local and remote XBee devices.
  - Profile update for local and remote XBee devices.
  - File system management for local XBee devices.
- New recover serial connection functionality to force the XBee serial connection settings.
- Support for notification of network cache modifications events (new node added, removed of existing node, network clear, ...)
- Deprecate get\_api\_output\_mode and set\_api\_output\_mode methods to use new
  get\_api\_output\_mode\_value and set\_api\_output\_mode\_value with APIOutputModeBit
  enumeration.
- Role as one of the cached parameters.
- Report an error on 'finished discovery' callback if node discovery fails.
- Several minor bug fixes.

#### 2.6.12.3 v1.2.0 - 04/05/2019

- Add new methods to send and receive data from other XBee interfaces through User Data Relay frames.
- Add new methods to manage the Bluetooth interface.
- Add support to set AT parameters without applying them with the AT Command Queue packet.
- Improve the callbacks mechanism:
  - Callbacks are now executed in parallel.
  - Internal callbacks are now defined when needed to avoid issues when more than one callback of the same type is defined.
- Add missing 'Transmit Status', 'Modem Status' and 'Cellular Association Indication Status' values to cover all XBee Cellular/XBee3 Cellular features.
- Bug Fixing:
  - Fix some bugs related to package spec data.
  - Log an error when processing a wrong frame instead of stopping the reader.
  - Fix an issue parsing Explicit RX Indicator packets.
  - Fix a couple of leaks with StreamHandlers.

### 2.6.12.4 v1.1.1 - 04/25/2018

- Add support for DigiMesh and 802.15.4 protocols on XBee3 modules.
- Return an unknown XBee packet when the received packet is not supported by the library instead of raising an
  exception.

- Change logging handler to log messages in the console.
- Bug Fixing:
  - Fix a problem when closing the device connection in the reader.
  - Fix how is determined whether the module has entered in AT command mode or not.
  - Fix the string encoding and decoding in some API packets.
  - Fix the message displayed when the XBee device protocol is not correct one.

### 2.6.12.5 v1.1.0 - 01/19/2018

- Add support for new hardware variants:
  - XB8X
- Add missing 'Modem Status' values for Remote Manager connect and disconnect events.
- Bug Fixing:
  - Fix timeouts on Unix platforms.
  - Fix the return source endpoint method from the 'ExplicitRXIndicatorPacket' class.
  - Perform general bug fixing when working in API escaped mode.

## 2.6.12.6 v1.0.0 - 10/02/2017

Initial release of XBee Python library. The main features of the library include:

- Support for ZigBee, 802.15.4, DigiMesh, Point-to-Multipoint, Wi-Fi, Cellular and NB-IoT devices.
- Support for API and API escaped operating modes.
- Management of local (attached to the PC) and remote XBee device objects.
- Discovery of remote XBee devices associated with the same network as the local device.
- Configuration of local and remote XBee devices:
  - Configure common parameters with specific setters and getters.
  - Configure any other parameter with generic methods.
  - Execute AT commands.
  - Apply configuration changes.
  - Write configuration changes.
  - Reset the device.
- Transmission of data to all the XBee devices on the network or to a specific device.
- Reception of data from remote XBee devices:
  - Data polling.
  - Data reception callback.
- · Transmission and reception of IP and SMS messages.
- Reception of network status changes related to the local XBee device.
- IO lines management:

- Configure IO lines.
- Set IO line value.
- Read IO line value.
- Receive IO data samples from any remote XBee device on the network.
- Support for explicit frames and application layer fields (Source endpoint, Destination endpoint, Profile ID, and Cluster ID).
- Multiple examples that show how to use the available APIs.

# 2.6.13 API reference

Following is API reference material on major parts of XBee Python library.

## 2.6.13.1 digi package

**Subpackages** 

digi.xbee package

**Subpackages** 

digi.xbee.models package

**Submodules** 

# digi.xbee.models.accesspoint module

Bases: object

This class represents an Access Point for the Wi-Fi protocol. It contains SSID, the encryption type and the link quality between the Wi-Fi module and the access point.

This class is used within the library to list the access points and connect to a specific one in the Wi-Fi protocol.

See also:

```
WiFiEncryptionType
```

Class constructor. Instantiates a new AccessPoint object with the provided parameters.

# **Parameters**

- ssid (String) the SSID of the access point.
- encryption\_type (WiFiEncryptionType) the encryption type configured in the access point.
- channel (Integer, optional) operating channel of the access point.

• signal\_quality (Integer, optional) - signal quality with the access point in %.

#### Raises

- ValueError if length of ssid is 0.
- ValueError if channel is less than 0.
- ValueError if *signal\_quality* is less than 0 or greater than 100.

### See also:

```
WiFiEncryptionType
```

## ssid

Returns the SSID of the access point.

**Returns** the SSID of the access point.

**Return type** String

# encryption\_type

Returns the encryption type of the access point.

**Returns** the encryption type of the access point.

Return type WiFiEncryptionType

#### See also:

```
WiFiEncryptionType
```

### channel

Returns the channel of the access point.

**Returns** the channel of the access point.

**Return type** Integer

See also:

```
AccessPoint.set channel()
```

# signal\_quality

Returns the signal quality with the access point in %.

**Returns** the signal quality with the access point in %.

Return type Integer

See also:

```
AccessPoint.__set_signal_quality()
```

```
class digi.xbee.models.accesspoint.WiFiEncryptionType (code, description)
     Bases: enum. Enum
          Enumerates the different Wi-Fi encryption types.
     Values:
           WiFiEncryptionType.NONE = (0, 'No security')
          WiFiEncryptionType.WPA = (1, 'WPA (TKIP) security')
          WiFiEncryptionType.WPA2 = (2, 'WPA2 (AES) security')
           WiFiEncryptionType.WEP = (3, 'WEP security')
     code
          Returns the code of the WiFiEncryptionType element.
              Returns the code of the WiFiEncryptionType element.
              Return type Integer
     description
          Returns the description of the WiFiEncryptionType element.
              Returns the description of the WiFiEncryptionType element.
              Return type String
digi.xbee.models.atcomm module
class digi.xbee.models.atcomm.ATStringCommand(command, description)
     Bases: enum. Enum
          This class represents basic AT commands.
          Inherited properties:
               name (String): name (ID) of this ATStringCommand.
               value (String): value of this ATStringCommand.
     Values:
          ATStringCommand.AC = ('AC', 'Apply changes')
          ATStringCommand.AG = ('AG', 'Aggregator support')
          ATStringCommand.AI = ('AI', 'Association indication')
          ATStringCommand.AO = ('AO', 'API options')
          ATStringCommand.AP = ('AP', 'API enable')
          ATStringCommand.AR = ('AR', 'Many-to-one route broadcast time')
          ATStringCommand.AS = ('AS', 'Active scan')
          ATStringCommand.BD = ('BD', 'UART baudrate')
          ATStringCommand.BI = ('BI', 'Bluetooth identifier')
          ATStringCommand.BL = ('BL', 'Bluetooth address')
          ATStringCommand.BP = ('BP', 'Bluetooth advertisement power')
          ATStringCommand.BT = ('BT', 'Bluetooth enable')
```

126

```
ATStringCommand.BR = ('BR', 'RF data rate')
ATStringCommand.C0 = ('C0', 'Source port')
ATStringCommand.C8 = ('C8', 'Compatibility mode')
ATStringCommand.CC = ('CC', 'Command sequence character')
ATStringCommand.CE = ('CE', 'Device role')
ATStringCommand.CH = ('CH', 'Channel')
ATStringCommand.CK = ('CK', 'Configuration checksum')
ATStringCommand.CM = ('CM', 'Channel mask')
ATStringCommand.CN = ('CN', 'Exit command mode')
ATStringCommand.DA = ('DA', 'Force Disassociation')
ATStringCommand.DB = ('DB', 'RSSI')
ATStringCommand.DD = ('DD', 'Device type')
ATStringCommand.DH = ('DH', 'Destination address high')
ATStringCommand.DJ = ('DJ', 'Disable joining')
ATStringCommand.DL = ('DL', 'Destination address low')
ATStringCommand.DM = ('DM', 'Disable device functionality')
ATStringCommand.DO = ('DO', 'Device options')
ATStringCommand.D0 = ('D0', 'DIO0 configuration')
ATStringCommand.D1 = ('D1', 'DIO1 configuration')
ATStringCommand.D2 = ('D2', 'DIO2 configuration')
ATStringCommand.D3 = ('D3', 'DIO3 configuration')
ATStringCommand.D4 = ('D4', 'DIO4 configuration')
ATStringCommand.D5 = ('D5', 'DIO5 configuration')
ATStringCommand.D6 = ('D6', 'RTS configuration')
ATStringCommand.D7 = ('D7', 'CTS configuration')
ATStringCommand.D8 = ('D8', 'DIO8 configuration')
ATStringCommand.D9 = ('D9', 'DIO9 configuration')
ATStringCommand.EE = ('EE', 'Encryption enable')
ATStringCommand.EO = ('EO', 'Encryption options')
ATStringCommand.FN = ('FN', 'Find neighbors')
ATStringCommand.FR = ('FR', 'Software reset')
ATStringCommand.FS = ('FS', 'File system')
ATStringCommand.GW = ('GW', 'Gateway address')
ATStringCommand.GT = ('GT', 'Guard times')
ATStringCommand.HV = ('HV', 'Hardware version')
ATStringCommand.HP = ('HP', 'Preamble ID')
ATStringCommand.IC = ('IC', 'Digital change detection')
ATStringCommand.ID = ('ID', 'Network PAN ID/Network ID/SSID')
ATStringCommand.IR = ('IR', 'I/O sample rate')
ATStringCommand.IS = ('IS', 'Force sample')
ATStringCommand.JN = ('JN', 'Join notification')
ATStringCommand.JV = ('JV', 'Join verification')
ATStringCommand.KY = ('KY', 'Link/Encryption key')
ATStringCommand.MA = ('MA', 'IP addressing mode')
ATStringCommand.MK = ('MK', 'IP address mask')
ATStringCommand.MP = ('MP', '16-bit parent address')
ATStringCommand.MY = ('MY', '16-bit address/IP address')
```

```
ATStringCommand.M0 = ('M0', 'PWM0 configuration')
ATStringCommand.M1 = ('M1', 'PWM1 configuration')
ATStringCommand.NB = ('NB', 'Parity')
ATStringCommand.NI = ('NI', 'Node identifier')
ATStringCommand.ND = ('ND', 'Node discover')
ATStringCommand.NJ = ('NJ', 'Join time')
ATStringCommand.NK = ('NK', 'Trust Center network key')
ATStringCommand.NO = ('NO', 'Node discover options')
ATStringCommand.NR = ('NR', 'Network reset')
ATStringCommand.NS = ('NS', 'DNS address')
ATStringCommand.NP = ('NP', 'Maximum number of transmission bytes')
ATStringCommand.NT = ('NT', 'Node discover back-off')
ATStringCommand.N_QUESTION = ('N?', 'Network discovery timeout')
ATStringCommand.OP = ('OP', 'Operating extended PAN ID')
ATStringCommand.OS = ('OS', 'Operating sleep time')
ATStringCommand.OW = ('OW', 'Operating wake time')
ATStringCommand.PK = ('PK', 'Passphrase')
ATStringCommand.PL = ('PL', 'TX power level')
ATStringCommand.PP = ('PP', 'Output power')
ATStringCommand.PS = ('PS', 'MicroPython auto start')
ATStringCommand.P0 = ('P0', 'DIO10 configuration')
ATStringCommand.P1 = ('P1', 'DIO11 configuration')
ATStringCommand.P2 = ('P2', 'DIO12 configuration')
ATStringCommand.P3 = ('P3', 'UART DOUT configuration')
ATStringCommand.P4 = ('P4', 'UART DIN configuration')
ATStringCommand.P5 = ('P5', 'DIO15 configuration')
ATStringCommand.P6 = ('P6', 'DIO16 configuration')
ATStringCommand.P7 = ('P7', 'DIO17 configuration')
ATStringCommand.P8 = ('P8', 'DIO18 configuration')
ATStringCommand.P9 = ('P9', 'DIO19 configuration')
ATStringCommand.RE = ('RE', 'Restore defaults')
ATStringCommand.RR = ('RR', 'XBee retries')
ATStringCommand.R QUESTION = ('R?', 'Region lock')
ATStringCommand.SB = ('SB', 'Stop bits')
ATStringCommand.SC = ('SC', 'Scan channels')
ATStringCommand.SD = ('SD', 'Scan duration')
ATStringCommand.SH = ('SH', 'Serial number high')
ATStringCommand.SI = ('SI', 'Socket info')
ATStringCommand.SL = ('SL', 'Serial number low')
ATStringCommand.SM = ('SM', 'Sleep mode')
ATStringCommand.SN = ('SN', 'Sleep count')
ATStringCommand.SO = ('SO', 'Sleep options')
ATStringCommand.SP = ('SP', 'Sleep time')
ATStringCommand.SS = ('SS', 'Sleep status')
ATStringCommand.ST = ('ST', 'Wake time')
ATStringCommand.TP = ('TP', 'Temperature')
```

128

Chapter 2. Contents

**ATStringCommand.VH** = ('VH', 'Bootloader version')

```
ATStringCommand.VR = ('VR', 'Firmware version')
          ATStringCommand.WR = ('WR', 'Write')
          ATStringCommand.DOLLAR_S = ('$S', 'SRP salt')
          ATStringCommand.DOLLAR_V = ('$V', 'SRP salt verifier')
          ATStringCommand.DOLLAR_W = ('$W', 'SRP salt verifier')
          ATStringCommand.DOLLAR_X = ('$X', 'SRP salt verifier')
          ATStringCommand.DOLLAR_Y = ('$Y', 'SRP salt verifier')
          ATStringCommand.PERCENT_C = ('%C', 'Hardware/software compatibility')
          ATStringCommand.PERCENT P = (\%P), 'Invoke bootloader')
          ATStringCommand.PERCENT_U = ('%U', 'Recover')
          ATStringCommand.PERCENT_V = ('%V', 'Supply voltage')
     command
          AT command alias
              Returns The AT command alias.
              Return type String
     description
          AT command description.
              Returns The AT command description.
              Return type String
class digi.xbee.models.atcomm.SpecialByte(code)
     Bases: enum. Enum
          Enumerates all the special bytes of the XBee protocol that must be escaped when working on API 2
          mode.
          Inherited properties:
               name (String): name (ID) of this SpecialByte.
               value (String): the value of this SpecialByte.
     Values:
          Special Byte.ESCAPE\_BYTE = 125
          SpecialByte.HEADER_BYTE = 126
          SpecialByte.XON_BYTE = 17
          SpecialByte.XOFF_BYTE = 19
     code
          Returns the code of the SpecialByte element.
              Returns the code of the SpecialByte element.
              Return type Integer
```

```
class digi.xbee.models.atcomm.ATCommand(command, parameter=None)
    Bases: object
```

This class represents an AT command used to read or set different properties of the XBee device.

AT commands can be sent directly to the connected device or to remote devices and may have parameters.

After executing an AT Command, an AT Response is received from the device.

Class constructor. Instantiates a new ATCommand object with the provided parameters.

### **Parameters**

- **command** (*String*) AT Command, must have length 2.
- parameter (String or Bytearray, optional) The AT parameter value. Defaults to None. Optional.

Raises ValueError – if command length is not 2.

### command

Returns the AT command.

**Returns** the AT command.

Return type String

# get\_parameter\_string()

Returns this ATCommand parameter as a String.

**Returns** this ATCommand parameter. *None* if there is no parameter.

Return type String

### parameter

Returns the AT command parameter.

### Returns

the AT command parameter. *None* if there is no parameter.

Return type Bytearray

```
class digi.xbee.models.atcomm.ATCommandResponse(command, response=None, status=<ATCommandStatus.OK: (0, 'Status OK')>)
```

Bases: object

This class represents the response of an AT Command sent by the connected XBee device or by a remote device after executing an AT Command.

Class constructor.

## **Parameters**

- **command** (ATCommand) The AT command that generated the response.
- response (bytearray, optional) The command response. Default to None.
- **status** (*ATCommandStatus*, optional) The AT command status. Default to ATCommandStatus.OK

#### command

Returns the AT command.

Returns the AT command.

Return type ATCommand

### response

Returns the AT command response.

**Returns** the AT command response.

**Return type** Bytearray

#### status

Returns the AT command response status.

**Returns** The AT command response status.

Return type ATCommandStatus

# digi.xbee.models.filesystem module

```
class digi.xbee.models.filesystem.FSCmdType (code, description)
    Bases: enum.Enum
```

This enumeration lists all the available file system commands.

# Inherited properties:

```
name (String): Name (id) of this FSCmdType. value (String): Value of this FSCmdType.
```

### Values:

```
Open/create file (1) = (1, 'Open/create file')
Close file (2) = (2, 'Close file')
Read file (3) = (3, 'Read file')
Write file (4) = (4, 'Write file')
File hash (8) = (8, 'File hash')
Create directory (16) = (16, 'Create directory')
Open directory (17) = (17, 'Open directory')
Close directory (18) = (18, 'Close directory')
Read directory (19) = (19, 'Read directory')
Get directory path ID (28) = (28, 'Get directory path ID')
Rename (33) = (33, 'Rename')
```

code

Returns the code of the file system command element.

**Stat filesystem (64)** = (64, 'Stat filesystem') **Format filesystem (79)** = (79, 'Format filesystem')

**Returns** Code of the file system command element.

Return type Integer

**Delete** (47) = (47, 'Delete')

# description

Returns the description of the file system command element.

**Returns** Description of the file system command element.

**Return type** Integer

```
class digi.xbee.models.filesystem.FSCmd(cmd_type, direction=0, status=None)
    Bases: object
```

This class represents a file system command.

Class constructor. Instantiates a new FSCmd object with the provided parameters.

### **Parameters**

- cmd\_type (FSCmdType or Integer) The command type.
- **direction** (Integer, optional, default=0) If this command is a request (0) or a response (1).
- **status** (*FSCommandStatus* or Integer) Status of the file system command execution. Only for response commands.

## Raises

- ValueError If *cmd\_type* is not an integer or a *FSCmdType*.
- ValueError If *cmd\_type* is invalid.

## See also:

```
FSCmdType
```

## type

Returns the command type.

```
Returns The command type.
```

Return type FSCmdType

### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

**Return type** Integer

### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

### See also:

```
FSCommandStatus
FSCmd.status_value()
```

### status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

# Return type Integer

#### See also:

```
FSCmd.status()
```

### output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

### to\_dict()

Returns a dictionary with all information of the command fields.

Returns Dictionary with all info of the command fields.

Return type Dictionary

## classmethod create\_cmd(raw, direction=0)

Creates a file system command with the given parameters. This method ensures that the FSCmd returned is valid and is well built (if not exceptions are raised).

#### **Parameters**

- raw (Bytearray) Bytearray to create the command.
- **direction** (Integer, optional, default=0) If this command is a request (0) or a response (1).

**Returns** The file system command created.

Return type FSCmd

Raises InvalidPacketException — If something is wrong with raw and the command cannot be built.

```
class digi.xbee.models.filesystem.UnknownFSCmd(raw, direction=0)
    Bases: digi.xbee.models.filesystem.FSCmd
```

buses. argr. Abec. moders. rrreby beck. r belief

This class represents an unknown file system command.

Class constructor. Instantiates a new UnknownFSCmd object with the provided parameters.

### **Parameters**

- ${\tt raw}\,({\it Bytearray})$  Data of the unknown command.
- **direction** (Integer, optional, default=0) If this command is a request (0) or a response (1).

**Raises** ValueError – If *data* is not a bytearray, its length is less than 3, or the command type is a known one.

See also:

**FSCmd** 

## type

Returns the command type.

**Returns** The command type.

Return type Integer

### classmethod create\_cmd(raw, direction=0)

Override method.

Returns UnknownFSCmd.

### **Raises**

- InvalidPacketException If raw is not a bytearray.
- InvalidPacketException If *raw* length is less than 3, or the command type is a known one.

## See also:

```
FSCmd.create_cmd()
```

## output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

### to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

Return type Dictionary

### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

**Return type** Integer

### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

### See also:

```
FSCommandStatus
FSCmd.status_value()
```

#### status value

Returns the file system command response status of the packet.

**Returns** File system command response status.

### Return type Integer

### See also:

```
FSCmd.status()
```

class digi.xbee.models.filesystem.FileIdCmd (cmd\_type, fid, direction=0, status=None)
 Bases: digi.xbee.models.filesystem.FSCmd

This class represents a file system command request or response that includes a file or path id.

Class constructor. Instantiates a new FileIdCmd object with the provided parameters.

#### **Parameters**

- cmd\_type (FSCmdType or Integer) The command type.
- **fid** (Integer) Id of the file/path to operate with. A file id expires and becomes invalid if not referenced for over 2 minutes. Set to 0x0000 for the root directory (/).
- **direction** (Integer, optional, default=0) If this command is a request (0) or a response (1).
- **status** (*FSCommandStatus* or Integer) Status of the file system command execution. Only for response commands.

**Raises** ValueError – If *fid* is invalid.

#### See also:

# FSCmd

**FSCommandStatus** 

# fs\_id

Returns the file/path identifier.

Returns The file/path id value.

Return type Integer

classmethod create\_cmd(raw, direction=0)

Override method.

Returns FileIdCmd.

**Raises** InvalidPacketException — If the bytearray length is less than the minimum required.

See also:

```
FSCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

```
Return type Integer
     output()
          Returns the raw bytearray of this command.
              Returns Raw bytearray of the command.
              Return type Bytearray
     status
          Returns the file system command response status.
              Returns File system command response status.
              Return type FSCommandStatus
          See also:
          FSCommandStatus
          FSCmd.status_value()
     status_value
          Returns the file system command response status of the packet.
              Returns File system command response status.
              Return type Integer
          See also:
          FSCmd.status()
     to dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
class digi.xbee.models.filesystem.FileIdNameCmd (cmd_type, fid, name, direction=0, sta-
                                                                tus=None)
     Bases: digi.xbee.models.filesystem.FileIdCmd
     This class represents a file system command request or response that includes a file or path id and a name.
     The file/path id is the next byte after the command type in the frame, and name are the following bytes until the
```

Class constructor. Instantiates a new FileIdNameCmd object with the provided parameters.

### **Parameters**

end of the frame.

• cmd\_type (FSCmdType or Integer) – The command type.

- **fid** (Integer) Id of the file/path to operate with. Set to 0x0000 for the root directory (/).
- name (String or bytearray) The path name of the file to operate with. Its maximum length is 252 characters.
- **direction** (Integer, optional, default=0) If this command is a request (0) or a response (1).
- **status** (*FSCommandStatus* or Integer) Status of the file system command execution. Only for response commands.

Raises ValueError – If fid or name are invalid.

### See also:

## **FSCmd**

#### name

Returns the path name of the file.

**Returns** The file path name.

Return type String

## classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns FileIdNameCmd.

**Raises** InvalidPacketException – If the bytearray length is less than the minimum required.

See also:

```
FSCmd.create_cmd()
```

### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

### fs\_id

Returns the file/path identifier.

**Returns** The file/path id value.

Return type Integer

### output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

```
Returns File system command response status.
```

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

#### status value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

#### to dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

Return type Dictionary

# type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

```
class digi.xbee.models.filesystem.OpenFileCmdRequest (path_id, name, flags)
    Bases: digi.xbee.models.filesystem.FileIdNameCmd
```

This class represents a file open/create file system command request. Open a file for reading and/or writing. Use *FileOpenRequestOption.SECURE* bitmask to upload a write-only file (one that cannot be downloaded or viewed), useful for protecting MicroPython source code on the device.

Command response is received as a OpenFileCmdResponse.

Class constructor. Instantiates a new OpenFileCmdRequest object with the provided parameters.

### **Parameters**

- path\_id (Integer) Directory path id. Set to 0x0000 for the root directory (/).
- name (String or bytearray) The path name of the file to open/create, relative to path\_id. Its maximum length is 251 chars.
- **flags** (FileOpenRequestOption) Bitfield of supported flags. Use FileOpenRequestOption to compose its value.

Raises ValueError – If any of the parameters is invalid.

See also:

```
FileIdNameCmd
FileOpenRequestOption
```

#### options

Returns the options to open the file.

**Returns** The options to open the file.

Return type FileOpenRequestOption

### classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns OpenFileCmdRequest.

### Raises

- InvalidPacketException If the bytearray length is less than 5. (cmd id + path id (2 bytes) + flags (1 byte) + name (at least 1 byte) = 5 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

#### See also:

```
FileIdNameCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

# fs\_id

Returns the file/path identifier.

Returns The file/path id value.

Return type Integer

#### name

Returns the path name of the file.

**Returns** The file path name.

Return type String

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

#### See also:

```
FSCommandStatus
FSCmd.status_value()
```

### status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

### to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

**Return type** Dictionary

# type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

```
class digi.xbee.models.filesystem.OpenFileCmdResponse(status, fid=None, size=None)
    Bases: digi.xbee.models.filesystem.FileIdCmd
```

This class represents a file open/create file system command response.

This is received in response of an OpenFileCmdRequest.

Class constructor. Instantiates a new OpenFileCmdResponse object with the provided parameters.

### **Parameters**

- status (FSCommandStatus or Integer) Status of the file system command execution.
- **fid** (Integer, optional, default=`None`) Id of the file that has been opened. It expires and becomes invalid if not referenced for over 2 minutes.
- size (Integer, optional, default=`None`) Size in bytes of the file. 0xFFFFFFF if unknown.

**Raises** ValueError – If any of the parameters is invalid.

# See also:

FileIdCmd

#### size

Returns the size of the opened file. 0xFFFFFFF if unknown.

```
Returns Size in bytes of the opened file.
```

**Return type** Integer

# classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns OpenFileCmdResponse.

#### Raises

- InvalidPacketException If the bytearray length is less than 8. (cmd id + status + file id (2 bytes) + size (4 bytes) = 8).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

#### See also:

```
FileIdCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### fs id

Returns the file/path identifier.

**Returns** The file/path id value.

Return type Integer

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

#### See also:

```
FSCommandStatus
FSCmd.status_value()
```

# status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

#### See also:

```
FSCmd.status()
```

#### to dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

Return type Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

```
class digi.xbee.models.filesystem.CloseFileCmdRequest(fid)
```

Bases: digi.xbee.models.filesystem.FileIdCmd

This class represents a file close file system command request. Close an open file and release its File Handle.

Command response is received as a CloseFileCmdResponse.

Class constructor. Instantiates a new CloseFileCmdRequest object with the provided parameters.

**Parameters fid** (Integer) – Id of the file to close returned in Open File Response. It expires and becomes invalid if not referenced for over 2 minutes.

Raises ValueError – If any of the parameters is invalid.

# See also:

FileIdCmd

# classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

 $\textbf{Returns} \ \textit{CloseFileCmdRequest}.$ 

#### Raises

- InvalidPacketException If the bytearray length is less than 3. (cmd id + file\_id (2 bytes) = 3 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

#### See also:

```
FileIdCmd.create_cmd()
```

# direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

```
Return type Integer
     fs_id
          Returns the file/path identifier.
              Returns The file/path id value.
              Return type Integer
     output()
          Returns the raw bytearray of this command.
              Returns Raw bytearray of the command.
              Return type Bytearray
     status
          Returns the file system command response status.
              Returns File system command response status.
              Return type FSCommandStatus
          See also:
          FSCommandStatus
          FSCmd.status_value()
     status_value
          Returns the file system command response status of the packet.
              Returns File system command response status.
              Return type Integer
          See also:
          FSCmd.status()
     to_dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
class digi.xbee.models.filesystem.CloseFileCmdResponse(status)
     Bases: digi.xbee.models.filesystem.FSCmd
     This class represents a file close file system command response.
     Command response is received as a CloseFileCmdRequest.
     Class constructor. Instantiates a new CloseFileCmdResponse object with the provided parameters.
```

```
Parameters status (FSCommandStatus or Integer) — Status of the file system command execution.

See also:
```

FSCmd

# classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns OpenFileCmdResponse.

#### **Raises**

- InvalidPacketException If the bytearray length is less than 1. (cmd id = 1 byte).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

### See also:

```
FSCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

**Return type** Integer

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

**Return type** Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

# status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

# to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

Return type Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

class digi.xbee.models.filesystem.ReadFileCmdRequest (fid, offset, size)

Bases: digi.xbee.models.filesystem.FileIdCmd

This class represents a read file system command request.

Command response is received as a ReadFileCmdResponse.

Class constructor. Instantiates a new ReadFileCmdRequest object with the provided parameters.

#### **Parameters**

- **fid** (Integer) Id of the file to read returned in Open File Response. It expires and becomes invalid if not referenced for over 2 minutes.
- **offset** (Integer) The file offset to start reading. 0xFFFFFFFF to use current position (ReadFileCmdRequest.USE\_CURRENT\_OFFSET)
- **size** (Integer) The number of bytes to read. 0xFFFF (ReadFileCm-dRequest.READ\_AS\_MANY) to read as many as possible (limited by file size or maximum response frame size)

Raises ValueError – If any of the parameters is invalid.

#### See also:

FileIdCmd

# USE\_CURRENT\_OFFSET = 4294967295

Use current file position to start reading.

#### $READ_AS_MANY = 65535$

Read as many bytes as possible (limited by file size or maximum response frame size)

# offset

Returns the file offset to start reading. 0xFFFFFFF to use current position (ReadFileCm-dRequest.0xFFFFFFFF)

**Returns** The file offset.

Return type Integer

# size

Returns the number of bytes to read. 0xFFFF (*ReadFileCmdRequest.READ\_AS\_MANY*) to read as many as possible (limited by file size or maximum response frame size)

```
Returns The number of bytes to read.
```

**Return type** Integer

# classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns ReadFileCmdRequest.

#### Raises

- InvalidPacketException If the bytearray length is less than 9. (cmd id + file\_id (2 bytes) + offset (4 bytes) + size (2 bytes) = 9 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

#### See also:

```
FileIdCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### fs id

Returns the file/path identifier.

**Returns** The file/path id value.

Return type Integer

#### output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

# status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

#### See also:

```
FSCmd.status()
```

#### to dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

Return type Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

Bases: digi.xbee.models.filesystem.FileIdCmd

This class represents a read file system command response.

Command response is received as a ReadFileCmdRequest.

Class constructor. Instantiates a new ReadFileCmdResponse object with the provided parameters.

#### **Parameters**

- status (FSCommandStatus or Integer) Status of the file system command execution.
- **fid**(Integer, optional, default=`None`) Id of the read file.
- offset (Integer, optional, default=`None`) The offset of the read data.
- data (Bytearray, optional, default=`None`) The file read data.

**Raises** ValueError – If any of the parameters is invalid.

# See also:

FileIdCmd

# offset

Returns the offset of the read data.

**Returns** The data offset.

Return type Integer

#### data

Returns the read data from the file.

Returns Read data.

Return type Bytearray

# classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns ReadFileCmdResponse.

### Raises

- InvalidPacketException If the bytearray length is less than 8. (cmd id + status + file\_id (2 bytes) + offset (4 bytes) + data = 8)
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

#### See also:

```
FileIdCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

### fs id

Returns the file/path identifier.

**Returns** The file/path id value.

Return type Integer

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

# status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

**Return type** Integer

See also:

```
FSCmd.status()
```

```
to dict()
```

Returns a dictionary with all information of the command fields.

Returns Dictionary with all info of the command fields.

**Return type** Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

class digi.xbee.models.filesystem.WriteFileCmdRequest(fid, offset, data=None)

Bases: digi.xbee.models.filesystem.FileIdCmd

This class represents a write file system command request.

Command response is received as a WriteFileCmdResponse.

Class constructor. Instantiates a new WriteFileCmdRequest object with the provided parameters.

#### **Parameters**

- **fid** (*Integer*) Id of the file to write returned in Open File Response. It expires and becomes invalid if not referenced for over 2 minutes.
- **offset** (Integer) The file offset to start writing. 0xFFFFFFF to use current position (ReadFileCmdRequest.USE\_CURRENT\_OFFSET)
- data (Bytearray, optional, default=`None`) The data to write. If empty, frame just refreshes the File Handle timeout to keep the file open.

Raises ValueError – If any of the parameters is invalid.

#### See also:

FileIdCmd

# USE\_CURRENT\_OFFSET = 4294967295

Use current file position to start writing.

#### offset

Returns the file offset to start writing.

**Returns** The file offset.

**Return type** Integer

#### data

Returns the data to write. If empty, frame just refreshes the File Handle timeout to keep the file open.

Returns The data to write.

Return type Bytearray

#### classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

**Returns** WriteFileCmdRequest.

Raises

- InvalidPacketException If the bytearray length is less than 7. (cmd id + file\_id (2 bytes) + offset (4 bytes) = 7 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

#### See also:

```
FileIdCmd.create_cmd()
```

### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### fs\_id

Returns the file/path identifier.

**Returns** The file/path id value.

Return type Integer

### output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

# status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

#### to\_dict()

150

Returns a dictionary with all information of the command fields.

Returns Dictionary with all info of the command fields.

Return type Dictionary

# type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

Bases: digi.xbee.models.filesystem.FileIdCmd

This class represents a write file system command response.

Command response is received as a WriteFileCmdRequest.

Class constructor. Instantiates a new WriteFileCmdResponse object with the provided parameters.

### **Parameters**

- **status** (*FSCommandStatus* or Integer) Status of the file system command execution.
- fid(Integer, optional, default=`None`) Id of the written file.
- actual\_offset (Integer, optional, default=`None`) The current file offset after writing.

Raises ValueError – If any of the parameters is invalid.

#### See also:

```
FileIdCmd
```

#### actual offset

Returns the file offset after writing.

**Returns** The file offset.

Return type Integer

classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns WriteFileCmdResponse.

#### **Raises**

- InvalidPacketException If the bytearray length is less than 8. (cmd id + status + file\_id (2 bytes) + offset (4 bytes) = 8)
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

# See also:

```
FileIdCmd.create_cmd()
```

direction

```
Returns the command direction.
         Returns 0 for request, 1 for response.
         Return type Integer
fs id
    Returns the file/path identifier.
         Returns The file/path id value.
         Return type Integer
output()
    Returns the raw bytearray of this command.
         Returns Raw bytearray of the command.
         Return type Bytearray
status
    Returns the file system command response status.
         Returns File system command response status.
         Return type FSCommandStatus
     See also:
     FSCommandStatus
    FSCmd.status_value()
status value
    Returns the file system command response status of the packet.
         Returns File system command response status.
         Return type Integer
     See also:
    FSCmd.status()
to dict()
     Returns a dictionary with all information of the command fields.
         Returns Dictionary with all info of the command fields.
         Return type Dictionary
type
     Returns the command type.
         Returns The command type.
         Return type FSCmdType
```

```
class digi.xbee.models.filesystem.HashFileCmdRequest(path_id, name)
    Bases: digi.xbee.models.filesystem.FileIdNameCmd
```

This class represents a file hash command request. Use this command to get a sha256 hash to verify a file's contents without downloading the entire file (something not even possible for secure files). On XBee Cellular modules, there is a response delay in order to calculate the hash of a non-secure file. Secure files on XBee Cellular and all files on XBee 3 802.15.4, DigiMesh, and Zigbee have a cached hash.

Command response is received as a <code>HashFileCmdResponse</code>.

Class constructor. Instantiates a new <code>HashFileCmdRequest</code> object with the provided parameters.

#### **Parameters**

- path\_id (Integer) Directory path id. Set to 0x0000 for the root directory (/).
- name (String or bytearray) The path name of the file to hash, relative to path\_id. Its maximum length is 252 chars.

Raises ValueError – If any of the parameters is invalid.

#### See also:

FileIdNameCmd

#### classmethod create cmd(raw, direction=0)

Override method. Direction must be 0.

Returns HashFileCmdRequest.

# Raises

- InvalidPacketException If the bytearray length is less than 4. (cmd id + path id (2 bytes) + name (at least 1 byte) = 4 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

# See also:

```
FileIdNameCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

### fs\_id

Returns the file/path identifier.

**Returns** The file/path id value.

**Return type** Integer

#### name

Returns the path name of the file.

**Returns** The file path name.

```
Return type String
     output()
          Returns the raw bytearray of this command.
              Returns Raw bytearray of the command.
              Return type Bytearray
     status
          Returns the file system command response status.
              Returns File system command response status.
              Return type FSCommandStatus
          See also:
          FSCommandStatus
          FSCmd.status_value()
     status_value
          Returns the file system command response status of the packet.
              Returns File system command response status.
              Return type Integer
          See also:
          FSCmd.status()
     to dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
class digi.xbee.models.filesystem.HashFileCmdResponse(status, file_hash=None)
     Bases: digi.xbee.models.filesystem.FSCmd
     This class represents a file hash command response.
     This is received in response of an HashFileCmdRequest.
     Class constructor. Instantiates a new <code>HashFileCmdResponse</code> object with the provided parameters.
          Parameters
                • status (FSCommandStatus or Integer) – Status of the file system command execution.
                • file_hash (Bytearray, optional, default=`None`) - The hash value.
```

```
Raises ValueError – If any of the parameters is invalid.
See also:
FSCmd
file hash
     Returns the hash of the file.
         Returns The hash of the file.
         Return type Bytearray
classmethod create_cmd(raw, direction=1)
     Override method. Direction must be 1.
         Returns HashFileCmdResponse.
         Raises
             • InvalidPacketException - If the bytearray length is less than 34. (cmd id + status
               + \text{ hash } (32 \text{ bytes}) = 34).
             • InvalidPacketException - If the command type is not FSCmdType or direction
               is not 1.
     See also:
     FSCmd.create_cmd()
direction
     Returns the command direction.
         Returns 0 for request, 1 for response.
         Return type Integer
output()
     Returns the raw bytearray of this command.
         Returns Raw bytearray of the command.
         Return type Bytearray
status
     Returns the file system command response status.
         Returns File system command response status.
```

2.6. API reference 155

Return type FSCommandStatus

See also:

FSCommandStatus
FSCmd.status\_value()

#### status value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

#### to dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

Return type Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

```
class digi.xbee.models.filesystem.CreateDirCmdRequest(path_id, name)
```

Bases: digi.xbee.models.filesystem.FileIdNameCmd

This class represents a create directory file system command request. Parent directories of the one to be created must exist. Separate request must be dane to make intermediate directories.

Command response is received as a CreateDirCmdResponse.

Class constructor. Instantiates a new CreateDirCmdRequest object with the provided parameters.

#### **Parameters**

- path\_id (Integer) Directory path id. Set to 0x0000 for the root directory (/).
- name (String or bytearray) The path name of the directory to create, relative to path\_id. Its maximum length is 252 chars.

**Raises** ValueError – If any of the parameters is invalid.

#### See also:

FileIdNameCmd

# classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns CreateDirCmdRequest.

#### Raises

- InvalidPacketException If the bytearray length is less than 4. (cmd id + path id (2 bytes) + name (at least 1 byte) = 4 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

# See also: FileIdNameCmd.create\_cmd() direction Returns the command direction. **Returns** 0 for request, 1 for response. Return type Integer fs id Returns the file/path identifier. Returns The file/path id value. Return type Integer name Returns the path name of the file. **Returns** The file path name. Return type String output() Returns the raw bytearray of this command. **Returns** Raw bytearray of the command. Return type Bytearray status Returns the file system command response status. **Returns** File system command response status. Return type FSCommandStatus See also: **FSCommandStatus** FSCmd.status\_value() status\_value Returns the file system command response status of the packet. **Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

#### to\_dict()

Returns a dictionary with all information of the command fields.

```
Returns Dictionary with all info of the command fields.
```

**Return type** Dictionary

# type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

# class digi.xbee.models.filesystem.CreateDirCmdResponse(status)

Bases: digi.xbee.models.filesystem.FSCmd

This class represents a create directory file system command response.

Command response is received as a CreateDirCmdRequest.

Class constructor. Instantiates a new CreateDirCmdResponse object with the provided parameters.

**Parameters** status (FSCommandStatus or Integer) – Status of the file system command execution.

#### See also:

**FSCmd** 

### classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns CreateDirCmdResponse.

# Raises

- InvalidPacketException If the bytearray length is less than 2. (cmd id + status = 2).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

# See also:

```
FSCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

### output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

```
Returns File system command response status.
              Return type FSCommandStatus
          See also:
          FSCommandStatus
          FSCmd.status_value()
     status value
          Returns the file system command response status of the packet.
              Returns File system command response status.
              Return type Integer
          See also:
          FSCmd.status()
     to dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
class digi.xbee.models.filesystem.OpenDirCmdRequest(path_id, name)
     Bases: digi.xbee.models.filesystem.FileIdNameCmd
     This class represents an open directory file system command request.
     Command response is received as a OpenDirCmdResponse.
     Class constructor. Instantiates a new OpenDirCmdRequest object with the provided parameters.
          Parameters
                • path_id (Integer) - Directory path id. Set to 0x0000 for the root directory (/).
                • name (String or bytearray) - Path name of the directory to open, relative to
                  path_id. An empty name is equivalent to '.', both refer to the current directory path id.
                  Its maximum length is 252 chars.
          Raises ValueError – If any of the parameters is invalid.
     See also:
     FileIdNameCmd
```

#### classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns OpenDirCmdRequest.

#### **Raises**

- InvalidPacketException If the bytearray length is less than 4. (cmd id + path id (2 bytes) + name (at least 1 byte) = 4 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

### See also:

```
FileIdNameCmd.create_cmd()
```

### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

# fs\_id

Returns the file/path identifier.

**Returns** The file/path id value.

Return type Integer

# name

Returns the path name of the file.

**Returns** The file path name.

Return type String

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

#### See also:

```
FSCommandStatus
FSCmd.status_value()
```

#### status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

### to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

**Return type** Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

Bases: digi.xbee.models.filesystem.FileIdCmd

This class represents an open directory file system command response. If the final file system element does not have *DirResponseFlag.ENTRY\_IS\_LAST* set, send a Directory Read Request to get additional entries. A response ending with an *DirResponseFlag.ENTRY\_IS\_LAST* flag automatically closes the Directory Handle. An empty directory returns a single entry with just the *DirResponseFlag.ENTRY\_IS\_LAST* flag set, and a 0-byte name.

This is received in response of an OpenDirCmdRequest.

Class constructor. Instantiates a new OpenFileCmdResponse object with the provided parameters.

#### **Parameters**

- status (FSCommandStatus or Integer) Status of the file system command execution.
- **did** (Integer, optional, default=`None`) Id of the directory that has been opened. It expires and becomes invalid if not referenced for over 2 minutes.
- **fs\_entries** (List, optional, default=`None`) List of bytearrays with the info and name of the entries inside the opened directory.

Raises ValueError – If any of the parameters is invalid.

#### See also:

```
FileIdCmd
```

#### is last

Returns whether there are more elements not included in this response.

#### Returns

True if there are no more elements to list, False otherwise.

Return type Boolean

#### fs entries

Returns the list of entries inside the opened directory.

**Returns** List of :class: .'FileSystemElement' inside the directory.

Return type List

#### classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns OpenDirCmdResponse.

#### Raises

- InvalidPacketException If the bytearray length is less than 8. (cmd id + status + dir id (2 bytes) + filesize\_and\_flags (4 bytes) = 8).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

### See also:

```
FileIdCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

# fs\_id

Returns the file/path identifier.

**Returns** The file/path id value.

Return type Integer

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

#### See also:

```
FSCommandStatus
FSCmd.status_value()
```

#### status\_value

162

Returns the file system command response status of the packet.

```
Returns File system command response status.
              Return type Integer
          See also:
          FSCmd.status()
     to_dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
class digi.xbee.models.filesystem.CloseDirCmdRequest(did)
     Bases: digi.xbee.models.filesystem.FileIdCmd
     This class represents a directory close file system command request.
     Command response is received as a CloseDirCmdResponse.
     Class constructor. Instantiates a new CloseDirCmdRequest object with the provided parameters.
          Parameters did (Integer) - Id of the directory to close. It expires and becomes invalid if not
              referenced for over 2 minutes.
          Raises ValueError – If any of the parameters is invalid.
     See also:
     FileIdCmd
     classmethod create_cmd(raw, direction=0)
          Override method. Direction must be 0.
              Returns CloseDirCmdRequest.
              Raises
                  • InvalidPacketException - If the bytearray length is less than 3. (cmd id + dir_id
                    (2 \text{ bytes}) = 3 \text{ bytes}).
                  • InvalidPacketException – If the command type is not FSCmdType or direction
```

See also:

```
FileIdCmd.create_cmd()
```

is not 0.

direction

```
Returns the command direction.
         Returns 0 for request, 1 for response.
         Return type Integer
fs id
    Returns the file/path identifier.
         Returns The file/path id value.
         Return type Integer
output()
    Returns the raw bytearray of this command.
         Returns Raw bytearray of the command.
         Return type Bytearray
status
    Returns the file system command response status.
         Returns File system command response status.
         Return type FSCommandStatus
     See also:
     FSCommandStatus
    FSCmd.status_value()
status value
    Returns the file system command response status of the packet.
         Returns File system command response status.
         Return type Integer
     See also:
    FSCmd.status()
to dict()
     Returns a dictionary with all information of the command fields.
         Returns Dictionary with all info of the command fields.
         Return type Dictionary
type
     Returns the command type.
         Returns The command type.
         Return type FSCmdType
```

```
class digi.xbee.models.filesystem.CloseDirCmdResponse(status)
    Bases: digi.xbee.models.filesystem.FSCmd
```

This class represents a directory close file system command response. Send this command to indicate that it is done reading the directory and no longer needs the Directory Handle. Typical usage scenario is to use a Directory Open Request and additional Directory Read Requests until the Response includes an entry with the *DirResponseFlag.ENTRY IS LAST* flag set.

Command response is received as a CloseDirCmdRequest.

Class constructor. Instantiates a new CloseDirCmdResponse object with the provided parameters.

**Parameters** status (FSCommandStatus or Integer) – Status of the file system command execution.

See also:

**FSCmd** 

#### classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns CloseDirCmdResponse.

#### **Raises**

- InvalidPacketException If the bytearray length is less than 2. (cmd id + status = 2).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

See also:

```
FSCmd.create_cmd()
```

# direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
          FSCmd.status_value()
     status_value
          Returns the file system command response status of the packet.
              Returns File system command response status.
              Return type Integer
          See also:
          FSCmd.status()
     to dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
class diqi.xbee.models.filesystem.ReadDirCmdRequest(did)
     Bases: digi.xbee.models.filesystem.FileIdCmd
     This class represents a directory read file system command request.
     Command response is received as a ReadDirCmdResponse.
     Class constructor. Instantiates a new ReadDirCmdRequest object with the provided parameters.
          Parameters did (Integer) – Id of the directory to close. It expires and becomes invalid if not
              referenced for over 2 minutes.
          Raises ValueError – If any of the parameters is invalid.
     See also:
     FileIdCmd
     classmethod create cmd(raw, direction=0)
          Override method. Direction must be 0.
              Returns ReadDirCmdRequest.
              Raises
                  • InvalidPacketException - If the bytearray length is less than 3. (cmd id + dir_id
```

• InvalidPacketException – If the command type is not FSCmdType or direction

(2 bytes) = 3 bytes).

is not 0.

# See also:

```
FileIdCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### fs id

Returns the file/path identifier.

Returns The file/path id value.

Return type Integer

#### output()

Returns the raw bytearray of this command.

Returns Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

#### status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

# to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

Return type Dictionary

### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

Bases: digi.xbee.models.filesystem.OpenDirCmdResponse

This class represents a read directory file system command response. If the final file system element does not have *DirResponseFlag.ENTRY\_IS\_LAST* set, send another Directory Read Request to get additional entries. A response ending with an *DirResponseFlag.ENTRY\_IS\_LAST* flag automatically closes the Directory Handle.

This is received in response of an ReadDirCmdRequest.

Class constructor. Instantiates a new ReadDirCmdResponse object with the provided parameters.

#### **Parameters**

- status (FSCommandStatus or Integer) Status of the file system command execution.
- **did** (Integer, optional, default=`None`) Id of the directory that has been read.
- **fs\_entries** (*List*, optional, default=`None`) List of bytearrays with the info and name of the entries inside the directory.

**Raises** ValueError – If any of the parameters is invalid.

#### See also:

FileIdCmd
DirResponseFlag

### classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns ReadDirCmdResponse.

# Raises

- InvalidPacketException If the bytearray length is less than 4. (cmd id + status + dir id (2 bytes) = 4).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

# See also:

```
FileIdCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### fs\_entries

Returns the list of entries inside the opened directory.

```
Return type List
fs_id
     Returns the file/path identifier.
         Returns The file/path id value.
         Return type Integer
is_last
     Returns whether there are more elements not included in this response.
         Returns
             True if there are no more elements to list, False otherwise.
         Return type Boolean
output()
     Returns the raw bytearray of this command.
         Returns Raw bytearray of the command.
         Return type Bytearray
status
     Returns the file system command response status.
         Returns File system command response status.
         Return type FSCommandStatus
     See also:
     FSCommandStatus
     FSCmd.status_value()
status_value
     Returns the file system command response status of the packet.
         Returns File system command response status.
         Return type Integer
     See also:
     FSCmd.status()
to_dict()
     Returns a dictionary with all information of the command fields.
         Returns Dictionary with all info of the command fields.
         Return type Dictionary
type
     Returns the command type.
         Returns The command type.
```

**Returns** List of :class: .'FileSystemElement' inside the directory.

#### Return type FSCmdType

class digi.xbee.models.filesystem.GetPathIdCmdRequest(path\_id, name)

Bases: digi.xbee.models.filesystem.FileIdNameCmd

This class represents a get path id file system command request. A directory path id (path\_id) of 0x0000 in any command, means path names are relative to the root directory of the filesystem (/).

- '/' as path separator
- '..' to refer to the parent directory
- '.' to refer to the current path directory

Use this command to get a shortcut to a subdirectory of the file system to allow the use of shorter path names in the frame:

- If the PATH ID field of this command is 0x0000, the XBee allocates a new PATH ID for use in later requests.
- If the PATH ID field of this command is non-zero, the XBee updates the directory path of that ID.

# To release a PATH ID when no longer needed:

- Send a request with that ID and a single slash ("/") as the pathname. Any Change Directory Request that resolves to the root directory releases the PATH ID and return a 0x0000 ID.
- Wait for a timeout (2 minutes)

Any file system id expires after 2 minutes if not referenced. Refresh this timeout by sending a Change Directory request with an empty or a single period ('.') as the pathname.

Command response is received as a GetPathIdCmdResponse.

Class constructor. Instantiates a new GetPathIdCmdRequest object with the provided parameters.

#### **Parameters**

- path\_id (Integer) Directory path id. Set to 0x0000 for the root directory (/).
- name (String or bytearray) The path name of the directory to change, relative to path\_id. An empty name is equivalent to '.', both refer to the current directory path id. Its maximum length is 252 chars.

Raises ValueError – If any of the parameters is invalid.

#### See also:

FileIdNameCmd

classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns GetPathIdCmdRequest.

# Raises

- InvalidPacketException If the bytearray length is less than 4. (cmd id + path id (2 bytes) + name (at least 1 byte) = 4 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

# See also: FileIdNameCmd.create\_cmd() direction Returns the command direction. **Returns** 0 for request, 1 for response. Return type Integer fs id Returns the file/path identifier. Returns The file/path id value. Return type Integer name Returns the path name of the file. **Returns** The file path name. Return type String output() Returns the raw bytearray of this command. **Returns** Raw bytearray of the command. Return type Bytearray status Returns the file system command response status. **Returns** File system command response status. Return type FSCommandStatus See also: **FSCommandStatus** FSCmd.status\_value() Returns the file system command response status of the packet.

# status\_value

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

#### to\_dict()

Returns a dictionary with all information of the command fields.

Returns Dictionary with all info of the command fields.

**Return type** Dictionary

# type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

Bases: digi.xbee.models.filesystem.FileIdCmd

This class represents a get path id file system command response. The full path of the new current directory is included if can fit.

This is received in response of an GetPathIdCmdRequest.

Class constructor. Instantiates a new GetPathIdCmdResponse object with the provided parameters.

#### **Parameters**

- $\bullet \ \ \textbf{status} \ (\textit{FSCommandStatus} \ or \ Integer) Status \ of \ the \ file \ system \ command \ execution.$
- path\_id(Integer, optional, default=`None`) New directory path id.
- **full\_path** (String or bytearray, optional, default=`None`) If short enough, the full path of the current directory, relative to path\_id. Deep subdirectories may return an empty field instead of their full path name. The maximum full path length is 255 characters.

**Raises** ValueError – If any of the parameters is invalid.

See also:

FileIdCmd

#### full path

Returns the full path of the current directory.

**Returns** The directory full path.

Return type String

classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns GetPathIdCmdResponse.

#### Raises

- InvalidPacketException If the bytearray length is less than 4. (cmd id + status + path id (2 bytes) = 4).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

See also:

FileIdNameCmd.create\_cmd()

```
Returns the command direction.
         Returns 0 for request, 1 for response.
         Return type Integer
fs id
    Returns the file/path identifier.
         Returns The file/path id value.
         Return type Integer
output()
    Returns the raw bytearray of this command.
         Returns Raw bytearray of the command.
         Return type Bytearray
status
    Returns the file system command response status.
         Returns File system command response status.
         Return type FSCommandStatus
     See also:
     FSCommandStatus
    FSCmd.status_value()
status value
    Returns the file system command response status of the packet.
         Returns File system command response status.
         Return type Integer
    See also:
    FSCmd.status()
to dict()
     Returns a dictionary with all information of the command fields.
         Returns Dictionary with all info of the command fields.
         Return type Dictionary
type
    Returns the command type.
         Returns The command type.
         Return type FSCmdType
```

direction

```
class digi.xbee.models.filesystem.RenameCmdRequest(path_id, name, new_name)
    Bases: digi.xbee.models.filesystem.FileIdNameCmd
```

This class represents a file/directory rename file system command request. Current firmware for XBee 3 802.15.4, DigiMesh, and Zigbee do not support renaming files. Contact Digi International to request it as a feature in a future release.

Command response is received as a RenameCmdResponse.

Class constructor. Instantiates a new RenameCmdRequest object with the provided parameters.

#### **Parameters**

- path\_id (Integer) Directory path id. Set to 0x0000 for the root directory (/).
- name (String or bytearray) The current path name of the file/directory to rename relative to path\_id. Its maximum length is 255 chars.
- new\_name (String or bytearray) The new name of the file/directory relative to path\_id. Its maximum length is 255 chars.

Raises ValueError – If any of the parameters is invalid.

#### See also:

FileIdNameCmd

#### new name

Returns the new name of the file or directory.

Returns The new name.

Return type String

# classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns RenameCmdRequest.

#### Raises

- InvalidPacketException If the bytearray length is less than 6. (cmd id + path id (2 bytes) + name (1 byte at least) + ',' + new name (at least 1 byte) = 6 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

# See also:

```
FileIdNameCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### fs id

Returns the file/path identifier.

```
Return type Integer
     name
          Returns the path name of the file.
              Returns The file path name.
              Return type String
     output()
          Returns the raw bytearray of this command.
              Returns Raw bytearray of the command.
              Return type Bytearray
     status
          Returns the file system command response status.
              Returns File system command response status.
              Return type FSCommandStatus
          See also:
          FSCommandStatus |
          FSCmd.status_value()
     status value
          Returns the file system command response status of the packet.
              Returns File system command response status.
              Return type Integer
          See also:
          FSCmd.status()
     to dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
class digi.xbee.models.filesystem.RenameCmdResponse(status)
     Bases: digi.xbee.models.filesystem.FSCmd
     This class represents a rename file system command response.
     Command response is received as a RenameCmdRequest.
```

**Returns** The file/path id value.

Class constructor. Instantiates a new RenameCmdResponse object with the provided parameters.

**Parameters** status (FSCommandStatus or Integer) – Status of the file system command execution.

## See also:

**FSCmd** 

# classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns RenameCmdResponse.

#### Raises

- InvalidPacketException If the bytearray length is less than 2. (cmd id + status = 2).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

## See also:

```
FSCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

#### output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

**Return type** Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

#### status value

176

Returns the file system command response status of the packet.

**Returns** File system command response status.

#### Return type Integer

#### See also:

```
FSCmd.status()
```

#### to dict()

Returns a dictionary with all information of the command fields.

Returns Dictionary with all info of the command fields.

**Return type** Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

```
class digi.xbee.models.filesystem.DeleteCmdRequest(path_id, name)
```

Bases: digi.xbee.models.filesystem.FileIdNameCmd

This class represents a delete file system command request. All files in a directory must be deleted before removing the directory. On XBee 3 802.15.4, DigiMesh, and Zigbee, deleted files are marked as as unusable space unless they are at the "end" of the file system (most-recently created). On these products, deleting a file triggers recovery of any deleted file space at the end of the file system, and can lead to a delayed response.

Command response is received as a DeleteCmdResponse.

Class constructor. Instantiates a new DeleteCmdRequest object with the provided parameters.

#### **Parameters**

- path\_id (Integer) Directory path id. Set to 0x0000 for the root directory (/).
- name (String or bytearray) The name of the file/directory to delete relative to path\_id. Its maximum length is 252 chars.

**Raises** ValueError – If any of the parameters is invalid.

#### See also:

FileIdNameCmd

## classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

Returns DeleteCmdRequest.

#### Raises

- InvalidPacketException If the bytearray length is less than 4. (cmd id + path id (2 bytes) + name (at least 1 byte) = 4 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

See also:

```
FileIdNameCmd.create cmd()
direction
    Returns the command direction.
         Returns 0 for request, 1 for response.
         Return type Integer
fs_id
     Returns the file/path identifier.
         Returns The file/path id value.
         Return type Integer
name
     Returns the path name of the file.
         Returns The file path name.
         Return type String
output()
    Returns the raw bytearray of this command.
         Returns Raw bytearray of the command.
         Return type Bytearray
status
    Returns the file system command response status.
         Returns File system command response status.
         Return type FSCommandStatus
     See also:
     FSCommandStatus
     FSCmd.status_value()
status_value
    Returns the file system command response status of the packet.
         Returns File system command response status.
         Return type Integer
     See also:
     FSCmd.status()
to_dict()
     Returns a dictionary with all information of the command fields.
         Returns Dictionary with all info of the command fields.
         Return type Dictionary
```

## type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

#### class digi.xbee.models.filesystem.DeleteCmdResponse(status)

Bases: digi.xbee.models.filesystem.FSCmd

This class represents a delete file system command response.

Command response is received as a DeleteCmdRequest.

Class constructor. Instantiates a new DeleteCmdResponse object with the provided parameters.

**Parameters** status (FSCommandStatus or Integer) – Status of the file system command execution.

## See also:

**FSCmd** 

## classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

Returns DeleteCmdResponse.

#### Raises

- InvalidPacketException If the bytearray length is less than 2. (cmd id + status = 2).
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

#### See also:

```
FSCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

**Return type** Integer

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

**Return type** Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

#### See also:

```
FSCommandStatus
FSCmd.status_value()
```

## status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

# to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

**Return type** Dictionary

# type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

```
class digi.xbee.models.filesystem.VolStatCmdRequest(name)
    Bases: digi.xbee.models.filesystem.FSCmd
```

This class represents a volume stat file system command request. Formatting the file system takes time, and any other requests fails until it completes and sends a response.

Command response is received as a VolStatCmdResponse.

Class constructor. Instantiates a new VolStatCmdRequest object with the provided parameters.

**Parameters name** (String or bytearray) – The name of the volume. Its maximum length is 254 characters.

Raises ValueError – If name is invalid.

See also:

**FSCmd** 

## name

Returns the name of the volume.

**Returns** The volume name.

**Return type** String

```
classmethod create_cmd(raw, direction=0)
```

Override method. Direction must be 0.

**Returns** VolStatCmdRequest.

#### **Raises**

- InvalidPacketException If the bytearray length is less than 2. (cmd id + name (at least 1 byte) = 2 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

## See also:

```
FSCmd.create_cmd()
```

## direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

## output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

# status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

#### to\_dict()

Returns a dictionary with all information of the command fields.

Returns Dictionary with all info of the command fields.

**Return type** Dictionary

# type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

Bases: digi.xbee.models.filesystem.FSCmd

This class represents a stat file system command response.

Command response is received as a VolStatCmdRequest.

Class constructor. Instantiates a new VolStatCmdResponse object with the provided parameters.

#### **Parameters**

- $\bullet \ \ \textbf{status} \ (\textit{FSCommandStatus} \ or \ Integer) \textbf{Status} \ of \ the \ file \ system \ command \ execution.$
- bytes\_used(Integer, optional, default=`None`)-Number of used bytes.
- bytes\_free (Integer, optional, default=`None`) Number of free bytes.
- bytes\_bad (Integer, optional, default=`None`) Number of bad bytes. For XBee 3 802.15.4, DigiMesh, and Zigbee, this represents space used by deleted files.

**Raises** ValueError – If any of the parameters is invalid.

## See also:

**FSCmd** 

## bytes\_used

Returns the used space on volume.

**Returns** Number of used bytes.

**Return type** Integer

#### bytes\_free

Returns the available space on volume.

**Returns** Number of free bytes.

Return type Integer

# bytes\_bad

Returns "bad" bytes on volume. For XBee 3 802.15.4, DigiMesh, and Zigbee, this represents space used by deleted files.

**Returns** Number of bad bytes.

Return type Integer

# classmethod create\_cmd(raw, direction=1)

Override method. Direction must be 1.

 $\textbf{Returns} \ \textit{VolStatCmdResponse}.$ 

# Raises

- InvalidPacketException If the bytearray length is less than 14. (cmd id + status + used (4 bytes) + free (4 bytes) + bad (4 bytes) = 14)
- InvalidPacketException If the command type is not FSCmdType or direction is not 1.

#### See also:

```
FileIdCmd.create_cmd()
```

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

## output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

#### status

Returns the file system command response status.

**Returns** File system command response status.

Return type FSCommandStatus

See also:

```
FSCommandStatus
FSCmd.status_value()
```

## status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
```

# to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

**Return type** Dictionary

# type Returns the command type. **Returns** The command type. Return type FSCmdType class digi.xbee.models.filesystem.VolFormatCmdRequest(name) Bases: digi.xbee.models.filesystem.VolStatCmdRequest This class represents a volume format file system command request. Command response is received as a VolFormatCmdResponse. Class constructor. Instantiates a new VolFormatCmdRequest object with the provided parameters. Parameters name (String or bytearray) - The name of the volume. Its maximum length is 254 chars. Raises ValueError - If name is invalid. See also: **FSCmd** direction Returns the command direction. **Returns** 0 for request, 1 for response. Return type Integer name Returns the name of the volume. **Returns** The volume name. **Return type** String output() Returns the raw bytearray of this command. **Returns** Raw bytearray of the command. Return type Bytearray status Returns the file system command response status. **Returns** File system command response status. Return type FSCommandStatus

#### See also:

```
FSCommandStatus
FSCmd.status_value()
```

#### status\_value

Returns the file system command response status of the packet.

```
Returns File system command response status.
```

Return type Integer

See also:

```
FSCmd.status()
```

## to\_dict()

Returns a dictionary with all information of the command fields.

**Returns** Dictionary with all info of the command fields.

**Return type** Dictionary

#### type

Returns the command type.

**Returns** The command type.

Return type FSCmdType

## classmethod create\_cmd(raw, direction=0)

Override method. Direction must be 0.

**Returns** VolFormatCmdRequest.

#### Raises

- InvalidPacketException If the bytearray length is less than 2. (cmd id + name (at least 1 byte) = 2 bytes).
- InvalidPacketException If the command type is not FSCmdType or direction is not 0.

#### See also:

```
FSCmd.create_cmd()
```

Bases: digi.xbee.models.filesystem.VolStatCmdResponse

This class represents a format file system command response.

Command response is received as a VolStatCmdRequest.

Class constructor. Instantiates a new VolFormatCmdResponse object with the provided parameters.

#### **Parameters**

- status (FSCommandStatus or Integer) Status of the file system command execution.
- bytes\_used(Integer, optional, default=`None`)-Number of used bytes.
- bytes\_free (Integer, optional, default=`None`) Number of free bytes.
- bytes\_bad(Integer, optional, default=`None`) Number of bad bytes.

Raises ValueError – If any of the parameters is invalid.

#### See also:

**FSCmd** 

# bytes\_bad

Returns "bad" bytes on volume. For XBee 3 802.15.4, DigiMesh, and Zigbee, this represents space used by deleted files.

Returns Number of bad bytes.

Return type Integer

## bytes\_free

Returns the available space on volume.

Returns Number of free bytes.

Return type Integer

## bytes\_used

Returns the used space on volume.

**Returns** Number of used bytes.

Return type Integer

#### direction

Returns the command direction.

**Returns** 0 for request, 1 for response.

Return type Integer

# output()

Returns the raw bytearray of this command.

**Returns** Raw bytearray of the command.

Return type Bytearray

# status

Returns the file system command response status.

**Returns** File system command response status.

 $\textbf{Return type} \ \textit{FSCommandStatus}$ 

See also:

```
FSCommandStatus
FSCmd.status_value()
```

# status\_value

Returns the file system command response status of the packet.

**Returns** File system command response status.

Return type Integer

See also:

```
FSCmd.status()
     to_dict()
          Returns a dictionary with all information of the command fields.
              Returns Dictionary with all info of the command fields.
              Return type Dictionary
     type
          Returns the command type.
              Returns The command type.
              Return type FSCmdType
     classmethod create_cmd(raw, direction=1)
          Override method. Direction must be 1.
              Returns VolFormatCmdResponse.
              Raises
                  • InvalidPacketException - If the bytearray length is less than 14. (cmd id + status
                    + used (4 bytes) + free (4 bytes) + bad (4 bytes) = 14)
                  • InvalidPacketException - If the command type is not FSCmdType or direction
                    is not 1.
          See also:
          FileIdCmd.create_cmd()
digi.xbee.models.hw module
class digi.xbee.models.hw.HardwareVersion(code, description)
     Bases: enum. Enum
          This class lists all hardware versions.
          Inherited properties:
               name (String): The name of this Hardware Version.
               value (Integer): The ID of this Hardware Version.
     Values:
          HardwareVersion.X09_009 = (1, 'X09-009')
          HardwareVersion.X09_019 = (2, 'X09-019')
          HardwareVersion.XH9_009 = (3, 'XH9-009')
          HardwareVersion.XH9_019 = (4, 'XH9-019')
          HardwareVersion.X24_009 = (5, 'X24-009')
          HardwareVersion.X24 019 = (6, 'X24-019')
          HardwareVersion.X09_001 = (7, 'X09-001')
```

2.6. API reference 187

**HardwareVersion.XH9\_001** = (8, 'XH9-001')

**HardwareVersion.X08 004** = (9, 'X08-004')

```
HardwareVersion.XC09_009 = (10, 'XC09-009')
HardwareVersion.XC09 038 = (11, 'XC09-038')
HardwareVersion.X24_038 = (12, 'X24-038')
HardwareVersion.X09_009_TX = (13, 'X09-009-TX')
HardwareVersion.X09 019 TX = (14, 'X09-019-TX')
HardwareVersion.XH9 009 TX = (15, 'XH9-009-TX')
HardwareVersion.XH9_019_TX = (16, 'XH9-019-TX')
HardwareVersion.X09 001 TX = (17, 'X09-001-TX')
HardwareVersion.XH9_001_TX = (18, 'XH9-001-TX')
HardwareVersion.XT09B XXX = (19, 'XT09B-xxx (Attenuator version)')
HardwareVersion.XT09_XXX = (20, 'XT09-xxx')
HardwareVersion.XC08_009 = (21, 'XC08-009')
HardwareVersion.XC08_038 = (22, 'XC08-038')
HardwareVersion.XB24 AXX XX = (23, 'XB24-Axx-xx')
HardwareVersion.XBP24_AXX_XX = (24, 'XBP24-Axx-xx')
HardwareVersion,XB24 BXIX XXX = (25, 'XB24-BxIx-xxx and XB24-Z7xx-xxx')
HardwareVersion.XBP24_BXIX_XXX = (26, 'XBP24-BxIx-xxx and XBP24-Z7xx-xxx')
HardwareVersion.XBP09 DXIX XXX = (27, 'XBP09-DxIx-xxx Digi Mesh')
HardwareVersion.XBP09_XCXX_XXX = (28, 'XBP09-XCxx-xxx: S3 XSC Compatibility')
HardwareVersion.XBP08 DXXX XXX = (29, 'XBP08-Dxx-xxx 868MHz')
HardwareVersion.XBP24B = (30, 'XBP24B: Low cost ZB PRO and PLUS S2B')
HardwareVersion.XB24_WF = (31, 'XB24-WF: XBee 802.11 (Redpine module)')
HardwareVersion.AMBER MBUS = (32, '??????: M-Bus module made by Amber')
HardwareVersion.XBP24C = (33, 'XBP24C: XBee PRO SMT Ember 357 S2C PRO')
HardwareVersion.XB24C = (34, 'XB24C: XBee SMT Ember 357 S2C')
HardwareVersion.XSC GEN3 = (35, 'XSC GEN3: XBP9 XSC 24 dBm')
HardwareVersion.SRD_868_GEN3 = (36, 'SDR_868_GEN3: XB8 12 dBm')
HardwareVersion.ABANDONATED = (37, 'Abandonated')
HardwareVersion.SMT 900LP = (38, "900LP (SMT): 900LP on 'S8 HW'")
HardwareVersion.WIFI ATHEROS = (39, 'WiFi Atheros (TH-DIP) XB2S-WF')
HardwareVersion.SMT_WIFI_ATHEROS = (40, 'WiFi Atheros (SMT) XB2B-WF')
HardwareVersion.SMT 475LP = (41, '475LP (SMT): Beta 475MHz')
HardwareVersion.XBEE CELL TH = (42, 'XBee-Cell (TH): XBee Cellular')
HardwareVersion.XLR MODULE = (43, 'XLR Module')
HardwareVersion.XB900HP_NZ = (44, 'XB900HP (New Zealand): XB9 NZ HW/SW')
HardwareVersion.XBP24C TH DIP = (45, 'XBP24C (TH-DIP): XBee PRO DIP')
HardwareVersion.XB24C_TH_DIP = (46, 'XB24C (TH-DIP): XBee DIP')
HardwareVersion.XLR BASEBOARD = (47, 'XLR Baseboard')
Hardware Version. XBP24C S2C SMT = (48, 'XBee PRO SMT')
HardwareVersion.SX PRO = (49, 'SX Pro')
HardwareVersion.S2D_SMT_PRO = (50, 'XBP24D: S2D SMT PRO')
HardwareVersion.S2D SMT REG = (51, 'XB24D: S2D SMT Reg')
HardwareVersion.S2D_TH_PRO = (52, 'XBP24D: S2D TH PRO')
HardwareVersion.S2D TH REG = (53, 'XB24D: S2D TH Reg')
HardwareVersion.SX = (62, 'SX')
HardwareVersion.XTR = (63, 'XTR')
```

```
HardwareVersion.CELLULAR CAT1 LTE VERIZON = (64, 'XBee Cellular Cat 1 LTE Verizon')
          HardwareVersion.XBEE3_SMT = (65, 'XBee 3 Micro and SMT')
          HardwareVersion.XBEE3_TH = (66, 'XBee 3 TH')
          HardwareVersion.XBEE3 = (67, 'XBee 3 Reserved')
          HardwareVersion.CELLULAR_3G = (68, 'XBee Cellular 3G')
          HardwareVersion.XB8X = (69, 'XB8X')
          HardwareVersion.CELLULAR_LTE_VERIZON = (70, 'XBee Cellular LTE-M Verizon')
          HardwareVersion.CELLULAR_LTE_ATT = (71, 'XBee Cellular LTE-M AT&T')
          HardwareVersion.CELLULAR NBIOT EUROPE = (72, 'XBee Cellular NBIoT Europe')
          HardwareVersion.CELLULAR_3_CAT1_LTE_ATT = (73, 'XBee Cellular 3 Cat 1 LTE AT&T')
          Hardware Version. CELLULAR 3 LTE M VERIZON = (74, 'XBee Cellular 3 LTE-M Verizon')
          HardwareVersion.CELLULAR_3_LTE_M_ATT = (75, 'XBee Cellular 3 LTE-M AT&T')
          HardwareVersion.CELLULAR_3_CAT1_LTE_VERIZON = (77, 'XBee Cellular 3 Cat 1 LTE
          Verizon')
     code
          Returns the code of the Hardware Version element.
              Returns the code of the Hardware Version element.
              Return type Integer
     description
          Returns the description of the Hardware Version element.
              Returns the description of the Hardware Version element.
              Return type String
class digi.xbee.models.hw.LegacyHardwareVersion(code, letter)
     Bases: enum. Enum
          This class lists all legacy hardware versions.
          Inherited properties:
               name (String): The name of this LegacyHardwareVersion.
               value (Integer): The ID of this LegacyHardwareVersion.
     Values:
          LegacyHardwareVersion.A = (1, 'A')
          LegacyHardwareVersion.B = (2, 'B')
          LegacyHardwareVersion.C = (3, 'C')
          LegacyHardwareVersion.D = (4, 'D')
          LegacyHardwareVersion.E = (5, 'E')
          LegacyHardwareVersion.F = (6, 'F')
          LegacyHardwareVersion.G = (7, 'G')
          LegacyHardwareVersion.H = (8, 'H')
          LegacyHardwareVersion.I = (9, 'I')
```

2.6. API reference 189

**LegacyHardwareVersion.J** = (10, 'J')

```
LegacyHardwareVersion.K = (11, 'K')
LegacyHardwareVersion.L = (12, 'L')
LegacyHardwareVersion.M = (13, 'M')
LegacyHardwareVersion.N = (14, 'N')
LegacyHardwareVersion.O = (15, 'O')
LegacyHardwareVersion.P = (16, 'P')
LegacyHardwareVersion.Q = (17, 'Q')
LegacyHardwareVersion.R = (18, 'R')
LegacyHardwareVersion.S = (19, 'S')
LegacyHardwareVersion.T = (20, 'T')
LegacyHardwareVersion.U = (21, 'U')
LegacyHardwareVersion.V = (22, 'V')
LegacyHardwareVersion.W = (23, 'W')
LegacyHardwareVersion.X = (24, 'X')
LegacyHardwareVersion.Y = (25, 'Y')
LegacyHardwareVersion.Z = (26, 'Z')
```

#### code

Returns the code of the LegacyHardwareVersion element.

**Returns** the code of the LegacyHardwareVersion element.

Return type Integer

#### letter

Returns the letter of the LegacyHardwareVersion element.

**Returns** the letter of the LegacyHardwareVersion element.

Return type String

## digi.xbee.models.info module

This class represents the information of an XBee socket:

- Socket ID.
- State.
- Protocol.
- Local port.
- · Remote port.
- · Remote address.

Class constructor. Instantiates a *SocketInfo* object with the given parameters.

#### **Parameters**

190

• **socket\_id** (*Integer*) – The ID of the socket.

- **state** (*SocketInfoState*) The state of the socket.
- **protocol** (*IPProtocol*) The protocol of the socket.
- local\_port (Integer) The local port of the socket.
- remote\_port (Integer) The remote port of the socket.
- remote\_address (String) The remote IPv4 address of the socket.

#### static create socket info(raw)

Parses the given bytearray data and returns a SocketInfo object.

**Parameters raw** (Bytearray) – received data from the SI command with a socket ID as argument.

#### Returns

The socket information, or *None* if the provided data is invalid.

Return type SocketInfo

## static parse\_socket\_list(raw)

Parses the given bytearray data and returns a list with the active socket IDs.

**Parameters** raw (Bytearray) – received data from the SI command.

#### Returns

list with the IDs of all active (open) sockets, or empty list if there is not any active socket.

**Return type** List

## socket\_id

Returns the ID of the socket.

**Returns** the ID of the socket.

Return type Integer

#### state

Returns the state of the socket.

**Returns** the state of the socket.

Return type SocketInfoState

# protocol

Returns the protocol of the socket.

**Returns** the protocol of the socket.

Return type IPProtocol

#### local\_port

Returns the local port of the socket. This is 0 unless the socket is explicitly bound to a port.

**Returns** the local port of the socket.

Return type Integer

#### remote\_port

Returns the remote port of the socket.

**Returns** the remote port of the socket.

Return type Integer

#### remote address

Returns the remote IPv4 address of the socket. This is 0.0.0.0 for an unconnected socket.

**Returns** the remote IPv4 address of the socket.

Return type String

# digi.xbee.models.mode module

This class represents all operating modes available.

# Inherited properties:

```
name (String): the name (id) of this OperatingMode. value (String): the value of this OperatingMode.
```

## Values:

```
OperatingMode.AT_MODE = (0, 'AT mode')
OperatingMode.API_MODE = (1, 'API mode')
OperatingMode.ESCAPED_API_MODE = (2, 'API mode with escaped characters')
OperatingMode.MICROPYTHON_MODE = (4, 'MicroPython REPL')
OperatingMode.BYPASS_MODE = (5, 'Bypass mode')
OperatingMode.UNKNOWN = (99, 'Unknown')
```

## code

Returns the code of the OperatingMode element.

**Returns** the code of the OperatingMode element.

Return type String

# description

Returns the description of the OperatingMode element.

**Returns** the description of the OperatingMode element.

Return type String

```
class digi.xbee.models.mode.APIOutputMode(code, description)
    Bases; enum.Enum
```

Enumerates the different API output modes. The API output mode establishes the way data will be output through the serial interface of an XBee device.

#### Inherited properties:

```
name (String): the name (id) of this OperatingMode. value (String): the value of this OperatingMode.
```

#### Values:

```
APIOutputMode.NATIVE = (0, 'Native')

APIOutputMode.EXPLICIT = (1, 'Explicit')

APIOutputMode.EXPLICIT_ZDO_PASSTHRU = (3, 'Explicit with ZDO Passthru')
```

#### code

Returns the code of the APIOutputMode element.

**Returns** the code of the APIOutputMode element.

Return type String

## description

Returns the description of the APIOutputMode element.

**Returns** the description of the APIOutputMode element.

Return type String

class digi.xbee.models.mode.APIOutputModeBit (code, description)

Bases: enum. Enum

Enumerates the different API output mode bit options. The API output mode establishes the way data will be output through the serial interface of an XBee.

# Inherited properties:

```
name (String): the name (id) of this APIOutputModeBit.value (String): the value of this APIOutputModeBit.
```

# Values:

**APIOutputModeBit.EXPLICIT** = (1, 'Output in Native/Explicit API format')

**APIOutputModeBit.SUPPORTED\_ZDO\_PASSTHRU** = (2, 'Zigbee: Supported ZDO request pass-throughn802.15.4/DigiMesh: Legacy API Indicator')

**APIOutputModeBit.UNSUPPORTED\_ZDO\_PASSTHRU** = (4, 'Unsupported ZDO request pass-through. Only Zigbee')

**APIOutputModeBit.BINDING\_PASSTHRU** = (8, 'Binding request pass-through. Only Zigbee') **APIOutputModeBit.ECHO\_RCV\_SUPPORTED\_ZDO** = (16, 'Echo received supported ZDO requests out the serial port. Only Zigbee')

**APIOutputModeBit.SUPPRESS\_ALL\_ZDO\_MSG** = (32, 'Suppress all ZDO messages from being sent out the serial port and disable pass-through. Only Zigbee')

#### code

Returns the code of the APIOutputModeBit element.

**Returns** the code of the APIOutputModeBit element.

Return type Integer

## description

Returns the description of the APIOutputModeBit element.

**Returns** the description of the APIOutputModeBit element.

Return type String

```
class digi.xbee.models.mode.IPAddressingMode(code, description)
```

Bases: enum. Enum

Enumerates the different IP addressing modes.

Values:

```
IPAddressingMode.DHCP = (0, 'DHCP')

IPAddressingMode.STATIC = (1, 'Static')
```

#### code

Returns the code of the IPAddressingMode element.

**Returns** the code of the IPAddressingMode element.

Return type String

# description

Returns the description of the IPAddressingMode element.

**Returns** the description of the IPAddressingMode element.

**Return type** String

```
class digi.xbee.models.mode.NeighborDiscoveryMode(code, description)
```

Bases: enum. Enum

Enumerates the different neighbor discovery modes. This mode establishes the way the network discovery process is performed.

Inherited properties:

```
name (String): the name (id) of this OperatingMode.value (String): the value of this OperatingMode.
```

Values:

```
NeighborDiscoveryMode.CASCADE = (0, 'Cascade')
NeighborDiscoveryMode.FLOOD = (1, 'Flood')
```

```
CASCADE = (0, 'Cascade')
```

The discovery of a node neighbors is requested once the previous request finishes. This means that just one discovery process is running at the same time.

This mode is recommended for large networks, it might be a slower method but it generates less traffic than 'Flood'.

```
FLOOD = (1, 'Flood')
```

The discovery of a node neighbors is requested when the node is found in the network. This means that several discovery processes might be running at the same time.

#### code

Returns the code of the NeighborDiscoveryMode element.

**Returns** the code of the NeighborDiscoveryMode element.

Return type String

#### description

Returns the description of the NeighborDiscoveryMode element.

**Returns** the description of the NeighborDiscoveryMode element.

**Return type** String

# digi.xbee.models.address module

```
class digi.xbee.models.address.XBee16BitAddress(address)
    Bases: object
```

This class represent a 16-bit network address.

This address is only applicable for:

- 1. 802.15.4
- 2. Zigbee
- 3. ZNet 2.5
- 4. XTend (Legacy)

DigiMesh and Point-to-multipoint does not support 16-bit addressing.

Each device has its own 16-bit address which is unique in the network. It is automatically assigned when the radio joins the network for Zigbee and Znet 2.5, and manually configured in 802.15.4 radios.

#### Attributes:

```
COORDINATOR_ADDRESS (XBee16BitAddress): 16-bit address reserved for the coordinator. BROADCAST_ADDRESS (XBee16BitAddress): 16-bit broadcast address. UNKNOWN_ADDRESS (XBee16BitAddress): 16-bit unknown address. PATTERN (String): Pattern for the 16-bit address string: (0[xX])?[0-9a-fA-F][1,4]
```

Class constructor. Instantiates a new XBeel 6BitAddress object with the provided parameters.

**Parameters address** (Bytearray) – address as byte array. Must be 1-2 digits.

#### Raises

- TypeError if address is None.
- ValueError if *address* is *None* or has less than 1 byte or more than 2.

```
PATTERN = '^(0[xX])?[0-9a-fA-F]{1,4}
```

16-bit address string pattern.

**Type** 16-bit address reserved for the coordinator (value

BROADCAST\_ADDRESS = <digi.xbee.models.address.XBee16BitAddress object>
 FFFF).

**Type** 16-bit broadcast address (value

UNKNOWN\_ADDRESS = <digi.xbee.models.address.XBee16BitAddress object>
 FFFE).

**Type** 16-bit unknown address (value

#### classmethod from hex string(address)

Class constructor. Instantiates a new :. XBee16BitAddress object from the provided hex string.

**Parameters address** (*String*) – String containing the address. Must be made by hex. digits without blanks. Minimum 1 character, maximum 4 (16-bit).

#### Raises

- ValueError if address has less than 1 character.
- ValueError if *address* contains non-hexadecimal characters.

#### classmethod from bytes (hsb, lsb)

Class constructor. Instantiates a new :. XBee16BitAddress object from the provided high significant byte and low significant byte.

#### **Parameters**

- **hsb** (*Integer*) high significant byte of the address.
- **1sb** (*Integer*) low significant byte of the address.

#### Raises

- ValueError if *lsb* is less than 0 or greater than 255.
- ValueError if *hsb* is less than 0 or greater than 255.

# classmethod is\_valid(address)

Checks if the provided hex string is a valid 16-bit address.

**Parameters address** (String or Bytearray, or XBee16BitAddress) – String: String with the address only with hex digits without blanks. Minimum 1 character, maximum 4 (16-bit). Bytearray: Address as byte array. Must be 1-2 digits.

**Returns** *True* for a valid 16-bit address, *False* otherwise.

Return type Boolean

## classmethod is\_known\_node\_addr(address)

Checks if a provided address is a known value. That is, if it is a valid 16-bit address and it is not the unknown or the broadcast address.

**Parameters address** (String, Bytearray, or XBee16BitAddress) – The 16-bit address to check as a string, bytearray or XBee16BitAddress.

**Returns** *True* for a known node 16-bit address, *False* otherwise.

Return type Boolean

#### get\_hsb()

Returns the high part of the bytearray (component 0).

**Returns** high part of the bytearray.

Return type Integer

#### get\_lsb()

Returns the low part of the bytearray (component 1).

**Returns** low part of the bytearray.

Return type Integer

#### address

Returns a bytearray representation of this XBee16BitAddress.

**Returns** bytearray representation of this XBee16BitAddress.

**Return type** Bytearray

class digi.xbee.models.address.XBee64BitAddress(address)

Bases: object

This class represents a 64-bit address (also known as MAC address).

The 64-bit address is a unique device address assigned during manufacturing. This address is unique to each physical device.

Class constructor. Instantiates a new XBee64BitAddress object with the provided parameters.

**Parameters** address (*Bytearray*) – the XBee 64-bit address as byte array.

**Raise:** ValueError: if *address* is *None* or its length less than 1 or greater than 8.

## PATTERN = $'^(0[xX])?[0-9a-fA-F]{1,16}$ \$'

64-bit address string pattern.

COORDINATOR\_ADDRESS = <digi.xbee.models.address.XBee64BitAddress object>
 00000000000000000.

**Type** 64-bit address reserved for the coordinator (value

BROADCAST\_ADDRESS = <digi.xbee.models.address.XBee64BitAddress object>
 000000000000FFFF).

**Type** 64-bit broadcast address (value

UNKNOWN\_ADDRESS = <digi.xbee.models.address.XBee64BitAddress object>
 FFFFFFFFFFFFF).

Type 64-bit unknown address (value

# classmethod from\_hex\_string(address)

Class constructor. Instantiates a new XBee64BitAddress object from the provided hex string.

**Parameters address** (String) – The XBee 64-bit address as a string.

**Raises** ValueError – if the address' length is less than 1 or does not match with the pattern: (0[xX])?[0-9a-fA-F]{1,16}.

# classmethod from\_bytes(\*args)

Class constructor. Instantiates a new XBee64BitAddress object from the provided bytes.

**Parameters args** (8 Integers) - 8 integers that represent the bytes 1 to 8 of this XBee64BitAddress.

**Raises** ValueError – if the amount of arguments is not 8 or if any of the arguments is not between 0 and 255.

#### classmethod is valid(address)

Checks if the provided hex string is a valid 64-bit address.

**Parameters address** (String, Bytearray, or XBee64BitAddress) – String: String with the address only with hex digits without blanks. Minimum 1 character, maximum 16 (64-bit). Bytearray: Address as byte array. Must be 1-8 digits.

**Returns** Boolean: *True* for a valid 64-bit address, *False* otherwise.

#### classmethod is known node addr(address)

Checks if a provided address is a known value. That is, if it is a valid 64-bit address and it is not the unknown or the broadcast address.

**Parameters address** (String, Bytearray, or XBee64BitAddress) – The 64-bit address to check as a string, bytearray or XBee64BitAddress.

**Returns** *True* for a known node 64-bit address, *False* otherwise.

Return type Boolean

#### address

Returns a bytearray representation of this XBee64BitAddress.

**Returns** bytearray representation of this XBee64BitAddress.

**Return type** Bytearray

class digi.xbee.models.address.XBeeIMEIAddress(address)

Bases: object

This class represents an IMEI address used by cellular devices.

This address is only applicable for Cellular protocol.

Class constructor. Instantiates a new :. XBeeIMEIAddress object with the provided parameters.

**Parameters** address (Bytearray) – The XBee IMEI address as byte array.

#### Raises

- ValueError if address is None.
- ValueError if length of address greater than 8.

## PATTERN = $'^{d{0,15}}$

IMEI address string pattern.

# classmethod from\_string(address)

Class constructor. Instantiates a new :. XBeeIMEIAddress object from the provided string.

**Parameters** address (String) – The XBee IMEI address as a string.

#### Raises

- ValueError if address is None.
- ValueError if *address* does not match the pattern: ^d{0,15}\$.

# classmethod is\_valid(address)

Checks if the provided hex string is a valid IMEI.

**Parameters address** (String or Bytearray) – The XBee IMEI address as a string or bytearray.

Returns True for a valid IMEI, False otherwise.

Return type Boolean

#### address

Returns a string representation of this XBeeIMEIAddress.

**Returns** the IMEI address in string format.

Return type String

# digi.xbee.models.message module

```
class digi.xbee.models.message.XBeeMessage(data, remote\_node, timestamp, broad-cast=False)
```

Bases: object

This class represents a XBee message, which is formed by a RemoteXBeeDevice (the sender) and some data (the data sent) as a bytearray.

Class constructor.

#### **Parameters**

- data (Bytearray) the data sent.
- remote\_node (RemoteXBeeDevice) the sender.
- **broadcast** (Boolean, optional, default=`False`) flag indicating whether the message is broadcast (*True*) or not (*False*). Optional.
- timestamp instant of time when the message was received.

#### data

Returns a bytearray containing the data of the message.

**Returns** the data of the message.

Return type Bytearray

#### remote device

Returns the device which has sent the message.

**Returns** the device which has sent the message.

Return type RemoteXBeeDevice

# is\_broadcast

Returns whether the message is broadcast or not.

**Returns** *True* if the message is broadcast, *False* otherwise.

Return type Boolean

#### timestamp

Returns the moment when the message was received as a time.time() function returned value.

#### Returns

the returned value of using time.time() function when the message was received.

Return type Float

## to\_dict()

Returns the message information as a dictionary.

Bases: digi.xbee.models.message.XBeeMessage

This class represents an Explicit XBee message, which is formed by all parameters of a common XBee message and: Source endpoint, destination endpoint, cluster ID, profile ID.

Class constructor.

#### **Parameters**

- data (Bytearray) the data sent.
- remote\_node (RemoteXBeeDevice) the sender device.
- **timestamp** instant of time when the message was received.
- **src endpoint** (*Integer*) source endpoint of the message. 1 byte.
- **dest\_endpoint** (*Integer*) destination endpoint of the message. 1 byte.
- **cluster\_id** (*Integer*) cluster id of the message. 2 bytes.
- **profile\_id** (*Integer*) **profile** id of the message. 2 bytes.
- **broadcast** (Boolean, optional, default=`False`) flag indicating whether the message is broadcast (*True*) or not (*False*). Optional.

## source\_endpoint

Returns the source endpoint of the message.

**Returns** the source endpoint of the message. 1 byte.

Return type Integer

# dest\_endpoint

Returns the destination endpoint of the message.

**Returns** the destination endpoint of the message. 1 byte.

Return type Integer

#### cluster id

Returns the cluster ID of the message.

**Returns** the cluster ID of the message. 2 bytes.

Return type Integer

# profile\_id

Returns the profile ID of the message.

**Returns** the profile ID of the message. 2 bytes.

Return type Integer

#### to dict()

Returns the message information as a dictionary.

# data

Returns a bytearray containing the data of the message.

**Returns** the data of the message.

Return type Bytearray

#### is broadcast

Returns whether the message is broadcast or not.

**Returns** *True* if the message is broadcast, *False* otherwise.

Return type Boolean

#### remote device

Returns the device which has sent the message.

**Returns** the device which has sent the message.

Return type RemoteXBeeDevice

#### timestamp

Returns the moment when the message was received as a time.time() function returned value.

#### Returns

the returned value of using time.time () function when the message was received.

# Return type Float

This class represents an IP message containing the IP address the message belongs to, the source and destination ports, the IP protocol, and the content (data) of the message.

Class constructor.

#### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address the message comes from.
- **src\_port** (*Integer*) TCP or UDP source port of the transmission.
- **dest\_port** (*Integer*) TCP or UDP destination port of the transmission.
- protocol (*IPProtocol*) IP protocol used in the transmission.
- data (Bytearray) the data sent.

## Raises

- ValueError if *ip\_addr* is *None*.
- ValueError if protocol is None.
- ValueError if data is None.
- ValueError if *source port* is less than 0 or greater than 65535.
- ValueError if *dest\_port* is less than 0 or greater than 65535.

# ip\_addr

Returns the IPv4 address this message is associated to.

**Returns** The IPv4 address this message is associated to.

Return type ipaddress. IPv4Address

## source\_port

Returns the source port of the transmission.

**Returns** The source port of the transmission.

Return type Integer

#### dest port

Returns the destination port of the transmission.

**Returns** The destination port of the transmission.

Return type Integer

#### protocol

Returns the protocol used in the transmission.

**Returns** The protocol used in the transmission.

Return type IPProtocol

#### data

Returns a bytearray containing the data of the message.

**Returns** the data of the message.

Return type Bytearray

#### to\_dict()

Returns the message information as a dictionary.

```
class digi.xbee.models.message.SMSMessage(phone_number, data)
    Bases: object
```

This class represents an SMS message containing the phone number that sent the message and the content (data) of the message.

This class is used within the library to read SMS sent to Cellular devices.

Class constructor. Instantiates a new SMSMessage object with the provided parameters.

# **Parameters**

- **phone\_number** (*String*) The phone number that sent the message.
- data (String) The message text.

#### Raises

- ValueError if *phone\_number* is *None*.
- ValueError if data is None.
- ValueError if *phone\_number* is not a valid phone number.

# phone\_number

Returns the phone number that sent the message.

**Returns** The phone number that sent the message.

**Return type** String

#### data

Returns the data of the message.

**Returns** The data of the message.

**Return type** String

# to\_dict()

Returns the message information as a dictionary.

This class represents a user data relay message containing the source interface and the content (data) of the message.

#### See also:

```
XBeeLocalInterface
```

Class constructor. Instantiates a new *UserDataRelayMessage* object with the provided parameters.

#### **Parameters**

- local\_iface (XBeeLocalInterface) The source XBee local interface.
- data (Bytearray) Byte array containing the data of the message.

**Raises** ValueError – if *relay\_interface* is *None*.

#### See also:

XBeeLocalInterface

# local\_interface

Returns the source interface that sent the message.

**Returns** The source interface that sent the message.

Return type XBeeLocalInterface

#### data

Returns the data of the message.

**Returns** The data of the message.

Return type Bytearray

# to\_dict()

Returns the message information as a dictionary.

# digi.xbee.models.options module

```
class digi.xbee.models.options.ReceiveOptions
    Bases: enum.Enum
```

This class lists all the possible options that have been set while receiving an XBee packet.

The receive options are usually set as a bitfield meaning that the options can be combined using the 'l' operand.

#### Values:

```
ReceiveOptions.NONE = 0
ReceiveOptions.PACKET_ACKNOWLEDGED = 1
ReceiveOptions.BROADCAST_PACKET = 2
ReceiveOptions.BROADCAST_PANS_PACKET = 4
ReceiveOptions.SECURE_SESSION_ENC = 16
```

ReceiveOptions.APS\_ENCRYPTED = 32
ReceiveOptions.SENT\_FROM\_END\_DEVICE = 64
ReceiveOptions.REPEATER\_MODE = 128
ReceiveOptions.DIGIMESH\_MODE = 192

#### NONE = 0

No special receive options.

#### PACKET ACKNOWLEDGED = 1

Packet was acknowledged.

Not valid for WiFi protocol.

#### BROADCAST PACKET = 2

Packet was sent as a broadcast.

Not valid for WiFi protocol.

#### BROADCAST PANS PACKET = 4

Packet was broadcast accros all PANs.

Only for 802.15.4 protocol.

#### SECURE SESSION ENC = 16

Packet sent across a Secure Session.

Only for XBee 3.

## $APS_ENCRYPTED = 32$

Packet encrypted with APS encryption.

Only valid for Zigbee protocol.

# SENT\_FROM\_END\_DEVICE = 64

Packet was sent from an end device (if known).

Only valid for Zigbee protocol.

#### POINT MULTIPOINT MODE = 64

Transmission is performed using point-to-Multipoint mode.

Only valid for DigiMesh 868/900 and Point-to-Multipoint 868/900 protocols.

# REPEATER\_MODE = 128

Transmission is performed using repeater mode.

Only valid for DigiMesh 868/900 and Point-to-Multipoint 868/900 protocols.

## DIGIMESH\_MODE = 192

Transmission is performed using DigiMesh mode.

Only valid for DigiMesh 868/900 and Point-to-Multipoint 868/900 protocols.

## class digi.xbee.models.options.TransmitOptions

Bases: enum. Enum

This class lists all the possible options that can be set while transmitting an XBee packet.

The transmit options are usually set as a bitfield meaning that the options can be combined using the 'l' operand.

Not all options are available for all cases, that's why there are different names with same values. In each moment, you must be sure that the option your are going to use, is a valid option in your context.

#### Values:

**TransmitOptions.NONE** = 0

**TransmitOptions.DISABLE\_ACK** = 1

TransmitOptions.DONT\_ATTEMPT\_RD = 2

TransmitOptions.USE\_BROADCAST\_PAN\_ID = 4

TransmitOptions.ENABLE\_MULTICAST = 8

**TransmitOptions.SECURE\_SESSION\_ENC** = 16

TransmitOptions.ENABLE\_APS\_ENCRYPTION = 32

**TransmitOptions.USE\_EXTENDED\_TIMEOUT = 64** 

**TransmitOptions.REPEATER\_MODE** = 128

**TransmitOptions.DIGIMESH\_MODE** = 192

#### NONE = 0

No special transmit options.

#### DISABLE ACK = 1

Disables acknowledgments on all unicasts.

Only valid for Zigbee, DigiMesh, 802.15.4, and Point-to-multipoint protocols.

# DISABLE\_RETRIES\_AND\_REPAIR = 1

Disables the retries and router repair in the frame.

Only valid for Zigbee protocol.

# DONT ATTEMPT RD = 2

Doesn't attempt Route Discovery.

Disables Route Discovery on all DigiMesh unicasts.

Only valid for DigiMesh protocol.

#### BROADCAST PAN = 2

Sends packet with broadcast {@code PAN ID}. Packet will be sent to all PANs.

Only valid for 802.15.4 XBee 3 protocol.

# $USE_BROADCAST_PAN_ID = 4$

Sends packet with broadcast {@code PAN ID}. Packet will be sent to all devices in the same channel ignoring the {@code PAN ID}.

It cannot be combined with other options.

Only valid for 802.15.4 XBee protocol.

## $ENABLE_UNICAST_NACK = 4$

Enables unicast NACK messages.

NACK message is enabled on the packet.

Only valid for DigiMesh 868/900 protocol, and XBee 3 DigiMesh.

#### ENABLE\_UNICAST\_TRACE\_ROUTE = 4

Enables unicast trace route messages.

Trace route is enabled on the packets.

Only valid for DigiMesh 868/900 protocol.

## INDIRECT TRANSMISSION = 4

Used for binding transmissions.

Only valid for Zigbee protocol.

#### ENABLE MULTICAST = 8

Enables multicast transmission request.

Only valid for Zigbee XBee protocol.

## ENABLE\_TRACE\_ROUTE = 8

Enable a unicast Trace Route on DigiMesh transmissions When set, the transmission will generate a Route Information - 0x8D frame.

Only valid for DigiMesh XBee protocol.

## SECURE\_SESSION\_ENC = 16

Encrypt payload for transmission across a Secure Session. Reduces maximum payload size by 4 bytes.

Only for XBee 3.

#### ENABLE APS ENCRYPTION = 32

Enables APS encryption, only if { @code EE=1 }.

Enabling APS encryption decreases the maximum number of RF payload bytes by 4 (below the value reported by {@code NP}).

Only valid for Zigbee XBee protocol.

## USE EXTENDED TIMEOUT = 64

Uses the extended transmission timeout.

Setting the extended timeout bit causes the stack to set the extended transmission timeout for the destination address.

Only valid for Zigbee XBee protocol.

## POINT\_MULTIPOINT\_MODE = 64

Transmission is performed using point-to-Multipoint mode.

Only valid for DigiMesh 868/900 and Point-to-Multipoint 868/900 protocols.

#### REPEATER MODE = 128

Transmission is performed using repeater mode.

Only valid for DigiMesh 868/900 and Point-to-Multipoint 868/900 protocols.

#### DIGIMESH\_MODE = 192

Transmission is performed using DigiMesh mode.

Only valid for DigiMesh 868/900 and Point-to-Multipoint 868/900 protocols.

#### class digi.xbee.models.options.RemoteATCmdOptions

Bases: enum. Enum

This class lists all the possible options that can be set while transmitting a remote AT Command.

These options are usually set as a bitfield meaning that the options can be combined using the 'l' operand.

Values:

```
RemoteATCmdOptions.NONE = 0
RemoteATCmdOptions.DISABLE_ACK = 1
RemoteATCmdOptions.APPLY_CHANGES = 2
RemoteATCmdOptions.SECURE_SESSION_ENC = 16
RemoteATCmdOptions.EXTENDED_TIMEOUT = 64
```

## NONE = 0

No special transmit options

#### DISABLE ACK = 1

Disables ACK

## $APPLY_CHANGES = 2$

Applies changes in the remote device.

If this option is not set, AC command must be sent before changes will take effect.

## SECURE SESSION ENC = 16

Send the remote command securely. Requires a Secure Session be established with the destination.

Only for XBee 3.

#### EXTENDED TIMEOUT = 64

Uses the extended transmission timeout.

Setting the extended timeout bit causes the stack to set the extended transmission timeout for the destination address.

Only valid for ZigBee XBee protocol.

```
class digi.xbee.models.options.SendDataRequestOptions(code, description)
```

Bases: enum. Enum

Enumerates the different options for the SendDataRequestPacket.

#### Values:

```
SendDataRequestOptions.OVERWRITE = (0, 'Overwrite')
SendDataRequestOptions.ARCHIVE = (1, 'Archive')
SendDataRequestOptions.APPEND = (2, 'Append')
SendDataRequestOptions.TRANSIENT = (3, 'Transient data (do not store)')
```

## code

Returns the code of the SendDataRequestOptions element.

**Returns** the code of the SendDataRequestOptions element.

Return type Integer

# description

Returns the description of the SendDataRequestOptions element.

**Returns** the description of the SendDataRequestOptions element.

Return type String

```
class digi.xbee.models.options.DiscoveryOptions (code, description)
     Bases: enum. Enum
          Enumerates the different options used in the discovery process.
     Values:
           DiscoveryOptions.APPEND_DD = (1, 'Append device type identifier (DD)')
          DiscoveryOptions.DISCOVER_MYSELF = (2, 'Local device sends response frame')
          DiscoveryOptions.APPEND_RSSI = (4, 'Append RSSI (of the last hop)')
     APPEND_DD = (1, 'Append device type identifier (DD)')
          Append device type identifier (DD) to the discovery response.
          Valid for the following protocols:

    DigiMesh

                • Point-to-multipoint (Digi Point)

    Zigbee

     DISCOVER_MYSELF = (2, 'Local device sends response frame')
          Local device sends response frame when discovery is issued.
          Valid for the following protocols:

    DigiMesh

                • Point-to-multipoint (Digi Point)
                • Zigbee
                • 802.15.4
     APPEND_RSSI = (4, 'Append RSSI (of the last hop)')
          Append RSSI of the last hop to the discovery response.
          Valid for the following protocols:
                • DigiMesh
                • Point-to-multipoint (Digi Point)
     code
          Returns the code of the DiscoveryOptions element.
              Returns the code of the DiscoveryOptions element.
              Return type Integer
     description
          Returns the description of the DiscoveryOptions element.
              Returns the description of the DiscoveryOptions element.
              Return type String
class digi.xbee.models.options.XBeeLocalInterface (code, description)
     Bases: enum. Enum
                              different
          Enumerates
                        the
                                         interfaces for
                                                            the
                                                                  UserDataRelayPacket
                                                                                               and
```

UserDataRelayOutputPacket.

#### Inherited properties:

**name** (String): the name (id) of the XBee local interface. **value** (String): the value of the XBee local interface.

#### Values:

**XBeeLocalInterface.SERIAL** = (0, 'Serial port (UART when in API mode, or SPI interface)')

**XBeeLocalInterface.BLUETOOTH** = (1, 'BLE API interface (on XBee devices which support BLE)')

**XBeeLocalInterface.MICROPYTHON** = (2, 'MicroPython')

**XBeeLocalInterface.UNKNOWN** = (255, 'Unknown interface')

#### code

Returns the code of the *XBeeLocalInterface* element.

**Returns** the code of the *XBeeLocalInterface* element.

Return type Integer

## description

Returns the description of the XBeeLocalInterface element.

**Returns** the description of the XBeeLocalInterface element.

Return type String

 $\verb"class" digi.xbee.models.options.RegisterKeyOptions" ( \textit{code}, \textit{description})$ 

Bases: enum. Enum

This class lists all the possible options that have been set while receiving an XBee packet.

The receive options are usually set as a bitfield meaning that the options can be combined using the 'l' operand.

#### Values:

**RegisterKeyOptions.LINK\_KEY** = (0, 'Key is a Link Key (KY on joining node)')

**RegisterKeyOptions.INSTALL\_CODE** = (1, 'Key is an Install Code (I? on joining node,DC must be set to 1 on joiner)')

**RegisterKeyOptions.UNKNOWN** = (255, 'Unknown key option')

#### code

Returns the code of the RegisterKeyOptions element.

**Returns** the code of the *RegisterKeyOptions* element.

**Return type** Integer

# description

Returns the description of the RegisterKeyOptions element.

**Returns** the description of the *RegisterKeyOptions* element.

**Return type** String

```
class digi.xbee.models.options.SocketOption(code, description)
     Bases: enum. Enum
          Enumerates the different Socket Options.
     Values:
          SocketOption.TLS_PROFILE = (0, 'TLS Profile')
          SocketOption.UNKNOWN = (255, 'Unknown')
     code
          Returns the code of the SocketOption element.
              Returns the code of the SocketOption element.
              Return type Integer
     description
          Returns the description of the SocketOption element.
              Returns the description of the SocketOption element.
              Return type String
class digi.xbee.models.options.FileOpenRequestOption
     Bases: enum. IntFlag
          This enumeration lists all the available options for FSCmdType.FILE_OPEN command requests.
          Inherited properties:
               name (String): Name (id) of this FileOpenRequestOption.
               value (String): Value of this FileOpenRequestOption.
     Values:
          FileOpenRequestOption.CREATE = 1
          FileOpenRequestOption.EXCLUSIVE = 2
          FileOpenRequestOption.READ = 4
          FileOpenRequestOption.WRITE = 8
          FileOpenRequestOption.TRUNCATE = 16
          FileOpenRequestOption.APPEND = 32
          FileOpenRequestOption.SECURE = 128
     CREATE = 1
          Create if file does not exist.
     EXCLUSIVE = 2
          Error out if file exists.
```

READ = 4

Open file for reading.

# WRITE = 8 Open fil

Open file for writing.

TRUNCATE = 16

Truncate file to 0 bytes.

APPEND = 32

Append to end of file.

SECURE = 128

Create a secure write-only file.

class digi.xbee.models.options.DirResponseFlag

Bases: enum. IntFlag

This enumeration lists all the available flags for FSCmdType.DIR\_OPEN and FSCmdType.DIR\_READ command responses.

Inherited properties:

name (String): Name (id) of this DirResponseFlag.value (String): Value of this DirResponseFlag.

Values:

DirResponseFlag.IS\_DIR = 128 DirResponseFlag.IS\_SECURE = 64 DirResponseFlag.IS\_LAST = 1

IS DIR = 128

Entry is a directory.

 $IS\_SECURE = 64$ 

Entry is stored securely.

 $IS\_LAST = 1$ 

Entry is the last.

# digi.xbee.models.protocol module

```
class digi.xbee.models.protocol.XBeeProtocol(code, description)
```

Bases: enum.Enum

Enumerates the available XBee protocols. The XBee protocol is determined by the combination of hardware and firmware of an XBee device.

Inherited properties:

```
name (String): the name (id) of this XBeeProtocol. value (String): the value of this XBeeProtocol.
```

Values:

**XBeeProtocol.ZIGBEE** = (0, 'Zigbee')

**XBeeProtocol.RAW\_802\_15\_4** = (1, '802.15.4')

```
XBeeProtocol.XBEE_WIFI = (2, 'Wi-Fi')
          XBeeProtocol.DIGI_MESH = (3, 'DigiMesh')
          XBeeProtocol.XCITE = (4, 'XCite')
          XBeeProtocol.XTEND = (5, 'XTend (Legacy)')
          XBeeProtocol.XTEND_DM = (6, 'XTend (DigiMesh)')
          XBeeProtocol.SMART_ENERGY = (7, 'Smart Energy')
          XBeeProtocol.DIGI_POINT = (8, 'Point-to-multipoint')
          XBeeProtocol.ZNET = (9, 'ZNet 2.5')
          XBeeProtocol.XC = (10, 'XSC')
          XBeeProtocol.XLR = (11, 'XLR')
          XBeeProtocol.XLR_DM = (12, 'XLR')
          XBeeProtocol.SX = (13, 'XBee SX')
          XBeeProtocol.XLR_MODULE = (14, 'XLR Module')
          XBeeProtocol.CELLULAR = (15, 'Cellular')
          XBeeProtocol.CELLULAR_NBIOT = (16, 'Cellular NB-IoT')
          XBeeProtocol.UNKNOWN = (99, 'Unknown')
     code
          Returns the code of the XBeeProtocol element.
              Returns the code of the XBeeProtocol element.
              Return type Integer
     description
          Returns the description of the XBeeProtocol element.
              Returns the description of the XBeeProtocol element.
              Return type String
class digi.xbee.models.protocol.IPProtocol(code, description)
     Bases: enum. Enum
          Enumerates the available network protocols.
          Inherited properties:
               name (String): the name (id) of this IPProtocol.
               value (String): the value of this IPProtocol.
     Values:
          IPProtocol.UDP = (0, 'UDP')
          IPProtocol.TCP = (1, 'TCP')
          IPProtocol.TCP\_SSL = (4, `TLS')
     code
          Returns the code of the IP protocol.
```

```
Return type Integer
     description
          Returns the description of the IP protocol.
               Returns description of the IP protocol.
               Return type String
class digi.xbee.models.protocol.Role(identifier, description)
     Bases: enum. Enum
          Enumerates the available roles for an XBee.
          Inherited properties:
                name (String): the name (id) of this Role.
                value (String): the value of this Role.
     Values:
           Role.COORDINATOR = (0, 'Coordinator')
           Role.ROUTER = (1, 'Router')
           Role.END_DEVICE = (2, 'End device')
           Role.UNKNOWN = (3, 'Unknown')
     id
          Gets the identifier of the role.
               Returns the role identifier.
               Return type Integer
     description
          Gets the description of the role.
               Returns the role description.
               Return type String
digi.xbee.models.status module
class digi.xbee.models.status.ATCommandStatus(code, description)
     Bases: enum. Enum
          This class lists all the possible states of an AT command after execution.
          Inherited properties:
                name (String): the name (id) of the ATCommandStatus.
                value (String): the value of the ATCommandStatus.
```

Values:

**Returns** code of the IP protocol.

```
ATCommandStatus.OK = (0, 'Status OK')
```

**ATCommandStatus.ERROR** = (1, 'Status Error')

ATCommandStatus.INVALID COMMAND = (2, 'Invalid command')

**ATCommandStatus.INVALID\_PARAMETER** = (3, 'Invalid parameter')

**ATCommandStatus.TX\_FAILURE** = (4, 'TX failure')

**ATCommandStatus.NO\_SECURE\_SESSION** = (11, 'No secure session: Remote command access requires a secure session be established first')

**ATCommandStatus.ENC\_ERROR** = (12, 'Encryption error')

**ATCommandStatus.CMD\_SENT\_INSECURELY** = (13, 'Command sent insecurely: A secure session exists, but the request needs to have the appropriate command option set (bit 4)')

**ATCommandStatus.UNKNOWN** = (255, 'Unknown status')

#### code

Returns the code of the ATCommandStatus element.

**Returns** the code of the ATCommandStatus element.

Return type Integer

## description

Returns the description of the ATCommandStatus element.

**Returns** the description of the ATCommandStatus element.

**Return type** String

class digi.xbee.models.status.DiscoveryStatus(code, description)

Bases: enum. Enum

This class lists all the possible states of the discovery process.

# Inherited properties:

```
name (String): The name of the DiscoveryStatus.value (Integer): The ID of the DiscoveryStatus.
```

## Values:

**DiscoveryStatus.NO\_DISCOVERY\_OVERHEAD** = (0, 'No discovery overhead')

**DiscoveryStatus.ADDRESS\_DISCOVERY** = (1, 'Address discovery')

**DiscoveryStatus.ROUTE\_DISCOVERY** = (2, 'Route discovery')

**DiscoveryStatus.ADDRESS\_AND\_ROUTE** = (3, 'Address and route')

**DiscoveryStatus.EXTENDED TIMEOUT DISCOVERY** = (64, 'Extended timeout discovery')

**DiscoveryStatus.UNKNOWN** = (255, 'Unknown')

## code

Returns the code of the DiscoveryStatus element.

**Returns** the code of the DiscoveryStatus element.

Return type Integer

#### description

Returns the description of the DiscoveryStatus element.

**Returns** The description of the DiscoveryStatus element.

Return type String

class digi.xbee.models.status.TransmitStatus(code, description)

Bases: enum. Enum

This class represents all available transmit status.

# Inherited properties:

**name** (String): the name (id) of ths TransmitStatus. **value** (String): the value of ths TransmitStatus.

#### Values:

**TransmitStatus.SUCCESS** = (0, 'Success')

**TransmitStatus.NO\_ACK** = (1, 'No acknowledgement received')

**TransmitStatus.CCA\_FAILURE** = (2, 'CCA failure')

**TransmitStatus.PURGED** = (3, 'Transmission purged, it was attempted before stack was up')

**TransmitStatus.WIFI\_PHYSICAL\_ERROR** = (4, 'Transceiver was unable to complete the transmission')

**TransmitStatus.INVALID\_DESTINATION** = (21, 'Invalid destination endpoint')

**TransmitStatus.NO\_BUFFERS** = (24, 'No buffers')

**TransmitStatus.NETWORK\_ACK\_FAILURE** = (33, 'Network ACK Failure')

**TransmitStatus.NOT JOINED NETWORK** = (34, 'Not joined to network')

**TransmitStatus.SELF\_ADDRESSED** = (35, 'Self-addressed')

TransmitStatus.ADDRESS\_NOT\_FOUND = (36, 'Address not found')

**TransmitStatus.ROUTE\_NOT\_FOUND** = (37, 'Route not found')

**TransmitStatus.BROADCAST\_FAILED** = (38, 'Broadcast source failed to hear a neighbor relay the message')

TransmitStatus.INVALID\_BINDING\_TABLE\_INDEX = (43, 'Invalid binding table index')

**TransmitStatus.INVALID\_ENDPOINT** = (44, 'Invalid endpoint')

TransmitStatus.BROADCAST\_ERROR\_APS = (45, 'Attempted broadcast with APS transmission')

**TransmitStatus.BROADCAST\_ERROR\_APS\_EE0** = (46, 'Attempted broadcast with APS transmission, but EE=0')

**TransmitStatus.SOFTWARE ERROR** = (49, 'A software error occurred')

TransmitStatus.RESOURCE\_ERROR = (50, 'Resource error lack of free buffers, timers, etc')

**TransmitStatus.NO\_SECURE\_SESSION** = (52, 'No Secure session connection')

**TransmitStatus.ENC\_FAILURE** = (53, 'Encryption failure')

**TransmitStatus.PAYLOAD\_TOO\_LARGE** = (116, 'Data payload too large')

TransmitStatus.INDIRECT\_MESSAGE\_UNREQUESTED = (117, 'Indirect message unrequested')

TransmitStatus.SOCKET\_CREATION\_FAILED = (118, 'Attempt to create a client socket failed')

**TransmitStatus.IP\_PORT\_NOT\_EXIST** = (119, 'TCP connection to given IP address and port does not exist. Source port is non-zero, so a new connection is not attempted')

**TransmitStatus.UDP\_SRC\_PORT\_NOT\_MATCH\_LISTENING\_PORT** = (120, 'Source port on a UDP transmission does not match a listening port on the transmitting module')

**TransmitStatus.TCP\_SRC\_PORT\_NOT\_MATCH\_LISTENING\_PORT** = (121, 'Source port on a TCP transmission does not match a listening port on the transmitting module')

TransmitStatus.INVALID\_IP\_ADDRESS = (122, 'Destination IPv4 address is invalid')

TransmitStatus.INVALID\_IP\_PROTOCOL = (123, 'Protocol on an IPv4 transmission is invalid')

**TransmitStatus.RELAY\_INTERFACE\_INVALID** = (124, 'Destination interface on a User Data Relay Frame does not exist')

**TransmitStatus.RELAY\_INTERFACE\_REJECTED** = (125, 'Destination interface on a User Data Relay Frame exists, but the interface is not accepting data')

**TransmitStatus.MODEM\_UPDATE\_IN\_PROGRESS** = (126, 'Modem update in progress. Try again after update completion.')

**TransmitStatus.SOCKET\_CONNECTION\_REFUSED** = (128, 'Destination server refused the connection')

**TransmitStatus.SOCKET\_CONNECTION\_LOST** = (129, 'The existing connection was lost before the data was sent')

TransmitStatus.SOCKET\_ERROR\_NO\_SERVER = (130, 'No server')

TransmitStatus.SOCKET\_ERROR\_CLOSED = (131, 'The existing connection was closed')

TransmitStatus.SOCKET\_ERROR\_UNKNOWN\_SERVER = (132, 'The server could not be found')

TransmitStatus.SOCKET\_ERROR\_UNKNOWN\_ERROR = (133, 'An unknown error occurred')

**TransmitStatus.INVALID\_TLS\_CONFIGURATION** = (134, 'TLS Profile on a 0x23 API request does not exist, or one or more certificates is invalid')

**TransmitStatus.SOCKET\_NOT\_CONNECTED** = (135, 'Socket not connected')

TransmitStatus.SOCKET\_NOT\_BOUND = (136, 'Socket not bound')

**TransmitStatus.KEY\_NOT\_AUTHORIZED** = (187, 'Key not authorized')

**TransmitStatus.UNKNOWN** = (255, 'Unknown')

#### code

Returns the code of the TransmitStatus element.

**Returns** the code of the TransmitStatus element.

**Return type** Integer

# description

Returns the description of the TransmitStatus element.

**Returns** the description of the TransmitStatus element.

**Return type** String

**class** digi.xbee.models.status.**ModemStatus**(code, description)

Bases: enum. Enum

Enumerates the different modem status events. This enumeration list is intended to be used within the  ${\tt ModemStatusPacket}$  packet.

## Values:

**ModemStatus.HARDWARE\_RESET** = (0, 'Device was reset')

**ModemStatus.WATCHDOG\_TIMER\_RESET** = (1, 'Watchdog timer was reset')

**ModemStatus.JOINED\_NETWORK** = (2, 'Device joined to network')

**ModemStatus.DISASSOCIATED** = (3, 'Device disassociated')

**ModemStatus.ERROR\_SYNCHRONIZATION\_LOST** = (4, 'Configuration error/synchronization lost')

**ModemStatus.COORDINATOR\_REALIGNMENT** = (5, 'Coordinator realignment')

**ModemStatus.COORDINATOR\_STARTED** = (6, 'The coordinator started')

**ModemStatus.NETWORK\_SECURITY\_KEY\_UPDATED** = (7, 'Network security key was updated')

**ModemStatus.NETWORK WOKE UP** = (11, 'Network woke up')

ModemStatus.NETWORK\_WENT\_TO\_SLEEP = (12, 'Network went to sleep')

ModemStatus.VOLTAGE\_SUPPLY\_LIMIT\_EXCEEDED = (13, 'Voltage supply limit exceeded')

ModemStatus.REMOTE\_MANAGER\_CONNECTED = (14, 'Remote Manager connected')

ModemStatus.REMOTE\_MANAGER\_DISCONNECTED = (15, 'Remote Manager disconnected')

**ModemStatus.MODEM\_CONFIG\_CHANGED\_WHILE\_JOINING** = (17, 'Modem configuration changed while joining')

ModemStatus.ACCESS\_FAULT = (18, 'Access fault')

**ModemStatus.FATAL\_ERROR** = (19, 'Fatal error')

**ModemStatus.BLUETOOTH\_CONNECTED** = (50, 'A Bluetooth connection has been made and API mode has been unlocked')

**ModemStatus.BLUETOOTH\_DISCONNECTED** = (51, 'An unlocked Bluetooth connection has been disconnected')

**ModemStatus.BANDMASK\_CONFIGURATION\_ERROR** = (52, 'LTE-M/NB-IoT bandmask configuration has failed')

**ModemStatus.CELLULAR\_UPDATE\_START** = (53, 'Cellular component update started')

ModemStatus.CELLULAR\_UPDATE\_FAILED = (54, 'Cellular component update failed')

ModemStatus.CELLULAR\_UPDATE\_SUCCESS = (55, 'Cellular component update completed')

ModemStatus.FIRMWARE\_UPDATE\_START = (56, 'XBee firmware update started')

ModemStatus.FIRMWARE UPDATE FAILED = (57, 'XBee firmware update failed')

ModemStatus.FIRMWARE\_UPDATE\_APPLYING = (58, 'XBee firmware update applying')

ModemStatus.SEC\_SESSION\_ESTABLISHED = (59, 'Secure session successfully established')

ModemStatus.SEC\_SESSION\_END = (60, 'Secure session ended')

ModemStatus.SEC SESSION AUTH FAILED = (61, 'Secure session authentication failed')

**ModemStatus.COORD\_PAN\_ID\_CONFLICT** = (62, 'Coordinator detected a PAN ID conflict but took no action because CR=0')

ModemStatus.COORD\_CHANGE\_PAN\_ID = (63, 'Coordinator changed PAN ID due to a conflict')

**ModemStatus.ROUTER\_PAN\_ID\_CHANGED** = (64, 'Router PAN ID was changed by coordinator due to a conflict')

**ModemStatus.NET\_WATCHDOG\_EXPIRED** = (66, 'Network watchdog timeout expired')

**ModemStatus.ERROR STACK** = (128, 'Stack error')

**ModemStatus.ERROR\_AP\_NOT\_CONNECTED** = (130, 'Send/join command issued without connecting from AP')

**ModemStatus.ERROR\_AP\_NOT\_FOUND** = (131, 'Access point not found')

**ModemStatus.ERROR\_PSK\_NOT\_CONFIGURED** = (132, 'PSK not configured')

**ModemStatus.ERROR\_SSID\_NOT\_FOUND** = (135, 'SSID not found')

ModemStatus.ERROR FAILED JOIN SECURITY = (136, 'Failed to join with security enabled')

**ModemStatus.ERROR\_INVALID\_CHANNEL** = (138, 'Invalid channel')

**ModemStatus.ERROR\_FAILED\_JOIN\_AP** = (142, 'Failed to join access point')

ModemStatus.UNKNOWN = (255, 'UNKNOWN')

#### code

Returns the code of the ModemStatus element.

**Returns** the code of the ModemStatus element.

**Return type** Integer

# description

Returns the description of the ModemStatus element.

**Returns** the description of the ModemStatus element.

Return type String

```
class digi.xbee.models.status.PowerLevel(code, description)
```

Bases: enum. Enum

Enumerates the different power levels. The power level indicates the output power value of a radio when transmitting data.

## Values:

**PowerLevel.LEVEL\_LOWEST** = (0, 'Lowest')

**PowerLevel.LEVEL\_LOW** = (1, 'Low')

**PowerLevel.LEVEL\_MEDIUM** = (2, 'Medium')

**PowerLevel.LEVEL\_HIGH** = (3, 'High')

**PowerLevel.LEVEL HIGHEST** = (4, 'Highest')

**PowerLevel.LEVEL\_UNKNOWN** = (255, 'Unknown')

#### code

Returns the code of the PowerLevel element.

Returns the code of the PowerLevel element.

Return type Integer

## description

Returns the description of the PowerLevel element.

**Returns** the description of the PowerLevel element.

**Return type** String

class digi.xbee.models.status.AssociationIndicationStatus(code, description)

Bases: enum.Enum

Enumerates the different association indication statuses.

# Values:

**AssociationIndicationStatus.SUCCESSFULLY\_JOINED** = (0, 'Successfully formed or joined a network')

**AssociationIndicationStatus.AS\_TIMEOUT** = (1, 'Active Scan Timeout')

AssociationIndicationStatus.AS\_NO\_PANS\_FOUND = (2, 'Active Scan found no PANs')

**AssociationIndicationStatus.AS\_ASSOCIATION\_NOT\_ALLOWED** = (3, 'Active Scan found PAN, but the CoordinatorAllowAssociation bit is not set')

**AssociationIndicationStatus.AS\_BEACONS\_NOT\_SUPPORTED** = (4, 'Active Scan found PAN, but Coordinator and End Device are not onfigured to support beacons')

**AssociationIndicationStatus.AS\_ID\_DOESNT\_MATCH** = (5, 'Active Scan found PAN, but the Coordinator ID parameter does not match the ID parameter of the End Device')

**AssociationIndicationStatus.AS\_CHANNEL\_DOESNT\_MATCH** = (6, 'Active Scan found PAN, but the Coordinator CH parameter does not match the CH parameter of the End Device')

AssociationIndicationStatus.ENERGY SCAN TIMEOUT = (7, 'Energy Scan Timeout')

**AssociationIndicationStatus.COORDINATOR\_START\_REQUEST\_FAILED** = (8, 'Coordinator start request failed')

**AssociationIndicationStatus.COORDINATOR\_INVALID\_PARAMETER** = (9, 'Coordinator could not start due to invalid parameter')

**AssociationIndicationStatus.COORDINATOR\_REALIGNMENT** = (10, 'Coordinator Realignment is in progress')

**AssociationIndicationStatus.AR\_NOT\_SENT** = (11, 'Association Request not sent')

**AssociationIndicationStatus.AR\_TIMED\_OUT** = (12, 'Association Request timed out - no reply was received')

**AssociationIndicationStatus.AR\_INVALID\_PARAMETER** = (13, 'Association Request had an Invalid Parameter')

**AssociationIndicationStatus.AR\_CHANNEL\_ACCESS\_FAILURE** = (14, 'Association Request Channel Access Failure. Request was not transmitted - CCA failure')

**AssociationIndicationStatus.AR\_COORDINATOR\_ACK\_WASNT\_RECEIVED** = (15, 'Remote Coordinator did not send an ACK after Association Request was sent')

**AssociationIndicationStatus.AR\_COORDINATOR\_DIDNT\_REPLY** = (16, 'Remote Coordinator did not reply to the Association Request, but an ACK was received after sending the request')

**AssociationIndicationStatus.SYNCHRONIZATION\_LOST** = (18, 'Sync-Loss - Lost synchronization with a Beaconing Coordinator')

**AssociationIndicationStatus.DISASSOCIATED** = (19, 'Disassociated - No longer associated to Coordinator')

**AssociationIndicationStatus.NO PANS FOUND** = (33, 'Scan found no PANs.')

**AssociationIndicationStatus.NO\_PANS\_WITH\_ID\_FOUND** = (34, 'Scan found no valid PANs based on current SC and ID settings')

**AssociationIndicationStatus.NJ\_EXPIRED** = (35, 'Valid Coordinator or Routers found, but they are not allowing joining (NJ expired)')

**AssociationIndicationStatus.NO\_JOINABLE\_BEACONS\_FOUND** = (36, 'No joinable beacons were found')

**AssociationIndicationStatus.UNEXPECTED\_STATE** = (37, 'Unexpected state, node should not be attempting to join at this time')

**AssociationIndicationStatus.JOIN\_FAILED** = (39, 'Node Joining attempt failed (typically due to incompatible security settings)')

**AssociationIndicationStatus.COORDINATOR\_START\_FAILED** = (42, 'Coordinator Start attempt failed')

**AssociationIndicationStatus.CHECKING\_FOR\_COORDINATOR** = (43, 'Checking for an existing coordinator')

**AssociationIndicationStatus.NETWORK\_LEAVE\_FAILED** = (44, 'Attempt to leave the network failed')

**AssociationIndicationStatus.DEVICE\_DIDNT\_RESPOND** = (171, 'Attempted to join a device that did not respond')

**AssociationIndicationStatus.UNSECURED\_KEY\_RECEIVED** = (172, 'Secure join error - network security key received unsecured')

**AssociationIndicationStatus.KEY\_NOT\_RECEIVED** = (173, 'Secure join error - network security key not received')

AssociationIndicationStatus.INVALID\_SECURITY\_KEY = (175, 'Secure join error - joining device

does not have the right preconfigured link key')

**AssociationIndicationStatus.SCANNING\_NETWORK** = (255, 'Scanning for a network/Attempting to associate')

## code

Returns the code of the AssociationIndicationStatus element.

**Returns** the code of the *AssociationIndicationStatus* element.

**Return type** Integer

# description

Returns the description of the AssociationIndicationStatus element.

#### Returns

the description of the AssociationIndicationStatus element.

Return type String

Bases: enum. Enum

Enumerates the different association indication statuses for the Cellular protocol.

## Values:

 $\label{lem:connected} \textbf{CellularAssociationIndicationStatus.SUCCESSFULLY\_CONNECTED} = (0, \text{`Connected to the Internet'})$ 

**Cellular Association Indication Status. REGISTERING\_CELLULAR\_NETWORK** = (34, 'Registering to cellular network')

**Cellular Association Indication Status. CONNECTING\_INTERNET** = (35, 'Connecting to the Internet')

**Cellular Association Indication Status. MODEM\_FIRMWARE\_CORRUPT** = (36, 'The cellular component requires a new firmware image')

**Cellular Association Indication Status. REGISTRATION\_DENIED** = (37, 'Cellular network registration was denied')

**CellularAssociationIndicationStatus.AIRPLANE\_MODE** = (42, 'Airplane mode is active')

CellularAssociationIndicationStatus.USB\_DIRECT = (43, 'USB Direct mode is active')

**Cellular Association Indication Status. PSM\_LOW\_POWER** = (44, 'The cellular component is in the PSM low-power state')

**CellularAssociationIndicationStatus.BYPASS\_MODE** = (47, 'Bypass mode active')

**Cellular Association Indication Status. INITIALIZING** = (255, 'Initializing')

#### code

Returns the code of the CellularAssociationIndicationStatus element.

## Returns

the code of the Cellular Association Indication Status element.

Return type Integer

## description

Returns the description of the Cellular Association Indication Status element.

#### Returns

the description of the Cellular Association Indication Status element.

Return type String

```
class digi.xbee.models.status.DeviceCloudStatus(code, description)
```

Bases: enum. Enum

Enumerates the different Device Cloud statuses.

#### Values:

**DeviceCloudStatus.SUCCESS** = (0, 'Success')

**DeviceCloudStatus.BAD\_REQUEST** = (1, 'Bad request')

**DeviceCloudStatus.RESPONSE\_UNAVAILABLE** = (2, 'Response unavailable')

**DeviceCloudStatus.DEVICE\_CLOUD\_ERROR** = (3, 'Device Cloud error')

**DeviceCloudStatus.CANCELED** = (32, 'Device Request canceled by user')

**DeviceCloudStatus.TIME\_OUT** = (33, 'Session timed out')

**DeviceCloudStatus.UNKNOWN ERROR** = (64, 'Unknown error')

#### code

Returns the code of the DeviceCloudStatus element.

**Returns** the code of the *DeviceCloudStatus* element.

Return type Integer

## description

Returns the description of the *DeviceCloudStatus* element.

**Returns** the description of the *DeviceCloudStatus* element.

Return type String

```
class digi.xbee.models.status.FrameError(code, description)
```

Bases: enum. Enum

Enumerates the different frame errors.

## Values:

**FrameError.INVALID\_TYPE** = (2, 'Invalid frame type')

**FrameError.INVALID\_LENGTH** = (3, 'Invalid frame length')

FrameError.INVALID\_CHECKSUM = (4, 'Erroneous checksum on last frame')

**FrameError.PAYLOAD\_TOO\_BIG** = (5, 'Payload of last API frame was too big to fit into a buffer')

**FrameError.STRING\_ENTRY\_TOO\_BIG** = (6, 'String entry was too big on last API frame sent')

FrameError.WRONG\_STATE = (7, 'Wrong state to receive frame')

**FrameError.WRONG\_REQUEST\_ID** = (8, 'Device request ID of device response do not match the number in the request')

#### code

Returns the code of the *FrameError* element.

**Returns** the code of the *FrameError* element.

Return type Integer

#### description

Returns the description of the FrameError element.

**Returns** the description of the *FrameError* element.

**Return type** String

class digi.xbee.models.status.WiFiAssociationIndicationStatus(code, tion)

Bases: enum. Enum

Enumerates the different Wi-Fi association indication statuses.

## Values:

**WiFiAssociationIndicationStatus.SUCCESSFULLY\_JOINED** = (0, 'Successfully joined to access point')

**WiFiAssociationIndicationStatus.INITIALIZING** = (1, 'Initialization in progress')

**WiFiAssociationIndicationStatus.INITIALIZED** = (2, 'Initialized, but not yet scanning')

**WiFiAssociationIndicationStatus.DISCONNECTING** = (19, 'Disconnecting from access point')

WiFiAssociationIndicationStatus.SSID\_NOT\_CONFIGURED = (35, 'SSID not configured')

**WiFiAssociationIndicationStatus.INVALID\_KEY** = (36, 'Encryption key invalid (NULL or invalid length)')

WiFiAssociationIndicationStatus.JOIN\_FAILED = (39, 'SSID found, but join failed')

**WiFiAssociationIndicationStatus.WAITING\_FOR\_AUTH** = (64, 'Waiting for WPA or WPA2 authentication')

**WiFiAssociationIndicationStatus.WAITING\_FOR\_IP** = (65, 'Joined to a network and waiting for IP address')

**WiFiAssociationIndicationStatus.SETTING\_UP\_SOCKETS** = (66, 'Joined to a network and IP configured. Setting up listening sockets')

**WiFiAssociationIndicationStatus.SCANNING\_FOR\_SSID** = (255, 'Scanning for the configured SSID')

## code

Returns the code of the WiFiAssociationIndicationStatus element.

**Returns** the code of the WiFiAssociationIndicationStatus element.

Return type Integer

## description

Returns the description of the WiFiAssociationIndicationStatus element.

**Returns** the description of the WiFiAssociationIndicationStatus element.

**Return type** String

 $\verb|class| \verb|digi.xbee.models.status|. \verb|NetworkDiscoveryStatus| (|code|, description|)$ 

Bases: enum. Enum

Enumerates the different statuses of the network discovery process.

# Values:

NetworkDiscoveryStatus.SUCCESS = (0, 'Success')

**NetworkDiscoveryStatus.ERROR\_READ\_TIMEOUT** = (1, 'Read timeout error')

**NetworkDiscoveryStatus.ERROR\_NET\_DISCOVER** = (2, 'Error executing node discovery')

**NetworkDiscoveryStatus.ERROR\_GENERAL** = (3, 'Error while discovering network')

**NetworkDiscoveryStatus.CANCEL** = (4, 'Discovery process cancelled')

#### code

Returns the code of the NetworkDiscoveryStatus element.

**Returns** the code of the *NetworkDiscoveryStatus* element.

Return type Integer

## description

Returns the description of the *NetworkDiscoveryStatus* element.

**Returns** the description of the *NetworkDiscoveryStatus* element.

Return type String

class digi.xbee.models.status.ZigbeeRegisterStatus(code, description)

Bases: enum. Enum

Enumerates the different statuses of the Zigbee Device Register process.

## Values:

```
ZigbeeRegisterStatus.SUCCESS = (0, 'Success')
```

**ZigbeeRegisterStatus.KEY\_TOO\_LONG** = (1, 'Key too long')

**ZigbeeRegisterStatus.ADDRESS\_NOT\_FOUND** = (177, 'Address not found in the key table')

**ZigbeeRegisterStatus.INVALID\_KEY** = (178, 'Key is invalid (00 and FF are reserved)')

**ZigbeeRegisterStatus.INVALID\_ADDRESS** = (179, 'Invalid address')

**ZigbeeRegisterStatus.KEY\_TABLE\_FULL** = (180, 'Key table is full')

**ZigbeeRegisterStatus.KEY\_NOT\_FOUND** = (255, 'Key not found')

**ZigbeeRegisterStatus.UNKNOWN** = (238, 'Unknown')

## code

Returns the code of the ZigbeeRegisterStatus element.

**Returns** the code of the *ZigbeeRegisterStatus* element.

Return type Integer

## description

Returns the description of the ZigbeeRegisterStatus element.

**Returns** the description of the *ZigbeeRegisterStatus* element.

Return type String

**Returns** the code of the *EmberBootloaderMessageType* element.

Return type Integer

#### description

Returns the description of the *EmberBootloaderMessageType* element.

**Returns** the description of the *EmberBootloaderMessageType* element.

Return type String

```
class digi.xbee.models.status.SocketStatus(code, description)
    Bases: enum.Enum
```

Enumerates the different Socket statuses.

## Values:

```
SocketStatus.SUCCESS = (0, 'Operation successful')

SocketStatus.INVALID_PARAM = (1, 'Invalid parameters')

SocketStatus.FAILED_TO_READ = (2, 'Failed to retrieve option value')

SocketStatus.CONNECTION_IN_PROGRESS = (3, 'Connection already in progress')

SocketStatus.ALREADY_CONNECTED = (4, 'Already connected/bound/listening')

SocketStatus.UNKNOWN_ERROR = (5, 'Unknown error')

SocketStatus.BAD_SOCKET = (32, 'Bad socket ID')

SocketStatus.NOT_REGISTERED = (34, 'Not registered to cell network')

SocketStatus.INTERNAL_ERROR = (49, 'Internal error')

SocketStatus.RESOURCE_ERROR = (50, 'Resource error: retry the operation later')

SocketStatus.INVALID_PROTOCOL = (123, 'Invalid protocol')

SocketStatus.UNKNOWN = (255, 'Unknown')
```

## code

Returns the code of the *SocketStatus* element.

```
Returns the code of the SocketStatus element.
              Return type Integer
     description
          Returns the description of the SocketStatus element.
              Returns the description of the SocketStatus element.
              Return type String
class digi.xbee.models.status.SocketState(code, description)
     Bases: enum. Enum
          Enumerates the different Socket states.
          SocketState.CONNECTED = (0, 'Connected')
          SocketState.FAILED_DNS = (1, 'Failed DNS lookup')
          SocketState.CONNECTION REFUSED = (2, 'Connection refused')
          SocketState.TRANSPORT_CLOSED = (3, 'Transport closed')
          SocketState.TIMED_OUT = (4, 'Timed out')
          SocketState.INTERNAL_ERROR = (5, 'Internal error')
          SocketState.HOST UNREACHABLE = (6, 'Host unreachable')
          SocketState.CONNECTION_LOST = (7, 'Connection lost')
          SocketState.UNKNOWN_ERROR = (8, 'Unknown error')
          SocketState.UNKNOWN_SERVER = (9, 'Unknown server')
          SocketState.RESOURCE ERROR = (10, 'Resource error')
          SocketState.LISTENER_CLOSED = (11, 'Listener closed')
          SocketState.UNKNOWN = (255, 'Unknown')
          Returns the code of the SocketState element.
              Returns the code of the SocketState element.
              Return type Integer
     description
          Returns the description of the SocketState element.
              Returns the description of the SocketState element.
              Return type String
class digi.xbee.models.status.SocketInfoState(code, description)
     Bases: enum. Enum
          Enumerates the different Socket info states.
```

Values:

code

Values:

2.6. API reference 225

**SocketInfoState.ALLOCATED** = (0, 'Allocated') **SocketInfoState.CONNECTING** = (1, 'Connecting') **SocketInfoState.CONNECTED** = (2, 'Connected')

```
SocketInfoState.LISTENING = (3, 'Listening')
           SocketInfoState.BOUND = (4, 'Bound')
           SocketInfoState.CLOSING = (5, 'Closing')
           SocketInfoState.UNKNOWN = (255, 'Unknown')
     code
           Returns the code of the SocketInfoState element.
               Returns the code of the SocketInfoState element.
               Return type Integer
     description
           Returns the description of the SocketInfoState element.
               Returns the description of the SocketInfoState element.
               Return type String
class digi.xbee.models.status.FSCommandStatus(code, description)
     Bases: enum. Enum
           This class lists all the possible states of an file system command after execution.
           Inherited properties:
                 name (String): Name (id) of the FSCommandStatus.
                 value (String): Value of the FSCommandStatus.
     Values:
           Success (0x00) = (0, 'Success')
           Error (0x01) = (1, 'Error')
           Invalid file system command (0x02) = (2, 'Invalid file system command')
           Invalid command parameter (0x03) = (3, 'Invalid command parameter')
           Access denied (0x50) = (80, 'Access denied')
           File or directory already exists (0x51) = (81, 'File or directory already exists')
           File or directory does not exist (0x52) = (82, \text{ 'File or directory does not exist'})
           Invalid file or directory name (0x53) = (83, 'Invalid file or directory name')
           File operation on directory (0x54) = (84, 'File operation on directory')
           Directory is not empty (0x55) = (85, 'Directory is not empty')
           Attempt to read past EOF (end of file) (0x56) = (86, \text{ 'Attempt to read past EOF (end of file)'})
           Hardware failure (0x57) = (87, 'Hardware failure')
            Volume offline / format required (0x58) = (88, \text{ 'Volume offline / format required'})
            Volume full (0x59) = (89, 'Volume full')
           Operation timed out (0x5A) = (90, 'Operation timed out')
           Busy with prior operation (0x5B) = (91, 'Busy with prior operation')
            Resource failure (memory allocation failed, try again) (0x5C) = (92, 'Resource failure (memory
```

allocation failed, try again)')

#### code

Returns the code of the FSCommandStatus element.

**Returns** Code of the FSCommandStatus element.

Return type Integer

## description

Returns the description of the FSCommandStatus element.

**Returns** Description of the FSCommandStatus element.

Return type String

## digi.xbee.models.zdo package

Bases: digi.xbee.models.zdo.\_ZDOCommand

This class performs a node descriptor read of the given XBee using a ZDO command.

The node descriptor read works only with Zigbee devices in API mode.

Class constructor. Instantiates a new NodeDescriptorReader object with the provided parameters.

#### **Parameters**

- (class (xbee) .XBeeDevice or class:.RemoteXBeeDevice): XBee to send the command.
- **configure\_ao** (Boolean, optional, default=`True`) True to set AO value before and after executing, False otherwise.
- timeout (Float, optional, default=`.\_\_DEFAULT\_TIMEOUT`) The ZDO command timeout in seconds.

#### Raises

- ValueError If xbee is None.
- ValueError If cluster\_id, receive\_cluster\_id, or timeout are less than 0.
- TypeError If the *xbee* is not a .*XBeeDevice* or a *RemoteXBeeDevice*.

# get\_node\_descriptor()

Returns the descriptor of the node.

**Returns** The node descriptor.

Return type NodeDescriptor

## error

Returns the error string if any.

**Returns** The error string.

Return type String

#### running

Returns if this ZDO command is running.

**Returns** *True* if it is running, *False* otherwise.

Return type Boolean

```
stop()
```

Stops the ZDO command process if it is running.

Bases: object

This class represents a node descriptor of an XBee.

Class constructor. Instantiates a new NodeDescriptor object with the provided parameters.

## **Parameters**

- role (Role) The device role.
- complex\_desc\_supported (Boolean) True if the complex descriptor is supported.
- user\_desc\_supported (Boolean) True if the user descriptor is supported.
- **freq\_band** (*List*) Byte array with the frequency bands.
- mac\_capabilities (List) Byte array with MAC capabilities.
- manufacturer\_code (Integer) The manufacturer's code assigned by the Zigbee Alliance.
- max\_buffer\_size (Integer) Maximum size in bytes of a data transmission.
- max\_in\_transfer\_size (Integer) Maximum number of bytes that can be received by the node.
- max\_out\_transfer\_size (Integer) Maximum number of bytes that can be transmitted by the node.
- **desc\_capabilities** (*List*) Byte array with descriptor capabilities.

#### role

Gets the role in this node descriptor.

**Returns** The role of the node descriptor.

Return type Role

See also:

Role

## complex\_desc\_supported

Gets if the complex descriptor is supported.

Returns True if supported, False otherwise.

Return type Boolean

## user\_desc\_supported

Gets if the user descriptor is supported.

**Returns** *True* if supported, *False* otherwise.

Return type Boolean

```
freq band
     868 MHz * Bit1: Reserved * Bit2: 900 MHz * Bit3: 2.4 GHz * Bit4: Reserved
         Returns List of integers with the frequency bands bits.
         Return type List
         Type Gets the frequency bands (LSB - bit0- index 0, MSB - bit4 - index 4)
         Type
             • Bit0
mac_capabilities
     Alternate PAN coordinator * Bit1: Device Type * Bit2: Power source * Bit3: Receiver on when idle *
     Bit4-5: Reserved * Bit6: Security capability * Bit7: Allocate address
         Returns List of integers with MAC capabilities bits.
         Return type List
         Type Gets the MAC capabilities (LSB - bit0- index 0, MSB - bit7 - index 7)
         Type
             · Bit0
manufacturer code
     Gets the manufacturer's code assigned by the Zigbee Alliance.
         Returns The manufacturer's code.
         Return type Integer
max_buffer_size
     Gets the maximum size in bytes of a data transmission (including APS bytes).
         Returns Maximum size in bytes.
         Return type Integer
max_in_transfer_size
     Gets the maximum number of bytes that can be received by the node.
         Returns Maximum number of bytes that can be received by the node.
         Return type Integer
max_out_transfer_size
     Gets the maximum number of bytes that can be transmitted by the node, including fragmentation.
         Returns Maximum number of bytes that can be transmitted by the node.
         Return type Integer
desc_capabilities
     Extended active endpoint list available * Bit1: Extended simple descriptor list available
         Returns List of integers with descriptor capabilities bits.
```

2.6. API reference 229

**Type** Gets the descriptor capabilities (LSB - bit0- index 0, MSB - bit1 - index 1)

**Return type** List

• Bit0

Type

```
class digi.xbee.models.zdo.RouteTableReader (xbee, configure_ao=True, timeout=20)
    Bases: digi.xbee.models.zdo._ZDOCommand
```

This class performs a route table read of the given XBee using a ZDO command.

The node descriptor read works only with Zigbee devices in API mode.

Class constructor. Instantiates a new RouteTableReader object with the provided parameters.

#### **Parameters**

- (class (xbee) .XBeeDevice or class:.RemoteXBeeDevice): XBee to send the command.
- **configure\_ao** (Boolean, optional, default=`True`) True to set AO value before and after executing, *False* otherwise.
- timeout (Float, optional, default=`.DEFAULT\_TIMEOUT`) The ZDO command timeout in seconds.

## Raises

- ValueError If xbee is None.
- ValueError If cluster\_id, receive\_cluster\_id, or timeout are less than 0.
- TypeError If the *xbee* is not a .*XBeeDevice* or a .*RemoteXBeeDevice*.

## get\_route\_table (route\_cb=None, finished\_cb=None)

Returns the routes of the XBee. If *route\_cb* is not defined, the process blocks until the complete routing table is read.

#### **Parameters**

- route\_cb (Function, optional, default=`None`) Method called when a new route is received. Receives two arguments:
  - The XBee that owns this new route.
  - The new route.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that executed the ZDO command.
  - A list with the discovered routes.
  - An error message if something went wrong.

#### Returns

**List of** *Route* **when** *route\_cb* **is not defined,** *None* **otherwise (in this case routes are received in the callback).** 

# Return type List

## See also:

Route

## error

Returns the error string if any.

**Returns** The error string.

## **Return type** String

#### running

Returns if this ZDO command is running.

**Returns** *True* if it is running, *False* otherwise.

**Return type** Boolean

stop()

Stops the ZDO command process if it is running.

class digi.xbee.models.zdo.RouteStatus(identifier, name)

Bases: enum. Enum

Enumerates the available route status.

id

Returns the identifier of the RouteStatus.

Returns RouteStatus identifier.

Return type Integer

Bases: object

This class represents a Zigbee route read from the route table of an XBee.

Class constructor. Instantiates a new Route object with the provided parameters.

## **Parameters**

- **destination** (XBee16BitAddress) 16-bit destination address of the route.
- next\_hop (XBee16BitAddress) 16-bit address of the next hop.
- **status** (*RouteStatus*) **Status** of the route.
- is\_low\_memory (Boolean) True to indicate if the device is a low-memory concentrator.
- **is\_many\_to\_one** (Boolean) *True* to indicate the destination is a concentrator.
- **is\_route\_record\_required** (Boolean) *True* to indicate a route record message should be sent prior to the next data transmission.

# See also:

RouteStatus XBee16BitAddress

## destination

Gets the 16-bit address of this route destination.

**Returns** 16-bit address of the destination.

Return type XBee16BitAddress

See also:

```
XBee16BitAddress
```

# next\_hop

Gets the 16-bit address of this route next hop.

**Returns** 16-bit address of the next hop.

Return type XBee16BitAddress

See also:

XBeel6BitAddress

## status

Gets this route status.

**Returns** The route status.

Return type RouteStatus

See also:

RouteStatus

# is\_low\_memory

Gets whether the device is a low-memory concentrator.

**Returns** *True* if the device is a low-memory concentrator, *False* otherwise.

Return type Boolean

## is\_many\_to\_one

Gets whether the destination is a concentrator.

**Returns** *True* if destination is a concentrator, *False* otherwise.

**Return type** Boolean

# is\_route\_record\_required

Gets whether a route record message should be sent prior the next data transmission.

**Returns** *True* if a route record message should be sent, *False* otherwise.

**Return type** Boolean

```
class digi.xbee.models.zdo.NeighborTableReader(xbee, configure_ao=True, timeout=20)
    Bases: digi.xbee.models.zdo._ZDOCommand
```

This class performs a neighbor table read of the given XBee using a ZDO command.

The node descriptor read works only with Zigbee devices in API mode.

Class constructor. Instantiates a new NeighborTableReader object with the provided parameters.

# **Parameters**

• (class(xbee) - .XBeeDevice or class:.RemoteXBeeDevice): XBee to send the command.

- **configure\_ao** (Boolean, optional, default=`True`) True to set AO value before and after executing, False otherwise.
- timeout (Float, optional, default=`.DEFAULT\_TIMEOUT`) The ZDO command timeout in seconds.

## Raises

- ValueError If xbee is None.
- ValueError If cluster\_id, receive\_cluster\_id, or timeout are less than 0.
- TypeError If the *xbee* is not a .*XBeeDevice* or a .*RemoteXBeeDevice*.

## get\_neighbor\_table (neighbor\_cb=None, finished\_cb=None)

Returns the neighbors of the XBee. If *neighbor\_cb* is not defined, the process blocks until the complete neighbor table is read.

## **Parameters**

- neighbor\_cb (Function, optional, default=`None`) Method called when a new neighbor is received. Receives two arguments:
  - The XBee that owns this new neighbor.
  - The new neighbor.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that executed the ZDO command.
  - A list with the discovered neighbors.
  - An error message if something went wrong.

## Returns

**List of** *Neighbor* **when** *neighbor\_cb* **is not defined,** *None* **otherwise (in this case neighbors are received in the callback)** 

Return type List

## See also:

Neighbor

#### error

Returns the error string if any.

**Returns** The error string.

Return type String

## running

Returns if this ZDO command is running.

**Returns** *True* if it is running, *False* otherwise.

Return type Boolean

# stop()

Stops the ZDO command process if it is running.

```
class digi.xbee.models.zdo.NeighborRelationship(identifier, name)
    Bases: enum.Enum
```

Dases. enum. Enum

Enumerates the available relationships between two nodes of the same network.

id

Returns the identifier of the NeighborRelationship.

**Returns** NeighborRelationship identifier.

Return type Integer

```
class digi.xbee.models.zdo.Neighbor (node, relationship, depth, lq)
Bases: object
```

This class represents a Zigbee or DigiMesh neighbor.

This information is read from the neighbor table of a Zigbee XBee, or provided by the 'FN' command in a Digimesh XBee.

Class constructor. Instantiates a new Neighbor object with the provided parameters.

#### Parameters

- node (RemoteXBeeDevice) The neighbor node.
- relationship (NeighborRelationship) The relationship of this neighbor with the node.
- **depth** (*Integer*) The tree depth of the neighbor. A value of 0 indicates the device is a Zigbee coordinator for the network. -1 means this is unknown.
- **lq** (*Integer*) The estimated link quality (LQI or RSSI) of data transmission from this neighbor.

# See also:

```
NeighborRelationship
RemoteXBeeDevice
```

#### node

Gets the neighbor node.

**Returns** The node itself.

Return type RemoteXBeeDevice

See also:

RemoteXBeeDevice

## relationship

Gets the neighbor node.

Returns The neighbor relationship.

Return type NeighborRelationship

See also:

## NeighborRelationship

# depth

Gets the tree depth of the neighbor.

**Returns** The tree depth of the neighbor.

**Return type** Integer

lq

Gets the estimated link quality (LQI or RSSI) of data transmission from this neighbor.

**Returns** The estimated link quality of data transmission from this neighbor.

Return type Integer

```
class digi.xbee.models.zdo.NeighborFinder(xbee, timeout=20)
    Bases: object
```

This class performs a find neighbors (FN) of an XBee. This action requires an XBee and optionally a find timeout.

The process works only in DigiMesh.

Class constructor. Instantiates a new NeighborFinder object with the provided parameters.

#### **Parameters**

- (class (xbee) .XBeeDevice or class:.RemoteXBeeDevice): The XBee to get neighbors from.
- timeout (Float) The timeout for the process in seconds.

## Raises

- OperationNotSupportedException If the process is not supported in the XBee.
- TypeError If the *xbee* is not a .*AbstractXBeeDevice*.
- ValueError If xbee is None.
- ValueError If timeout is less than 0.

## running

Returns whether this find neighbors process is running.

**Returns** *True* if it is running, *False* otherwise.

Return type Boolean

#### error

Returns the error string if any.

**Returns** The error string.

Return type String

#### stop()

Stops the find neighbors process if it is running.

```
get_neighbors (neighbor_cb=None, finished_cb=None)
```

Returns the neighbors of the XBee. If *neighbor\_cb* is not defined, the process blocks until the complete neighbor table is read.

#### **Parameters**

- neighbor\_cb (Function, optional, default=`None`) Method called when a new neighbor is received. Receives two arguments:
  - The XBee that owns this new neighbor.
  - The new neighbor.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that executed the FN command.
  - A list with the discovered neighbors.
  - An error message if something went wrong.

#### Returns

**List of** *Neighbor* **when** *neighbor\_cb* **is not defined,** *None* **otherwise (in this case neighbors are received in the callback)** 

**Return type** List

See also:

Neighbor

## digi.xbee.packets package

## **Submodules**

# digi.xbee.packets.aft module

```
class digi.xbee.packets.aft.ApiFrameType(code, description)
    Bases: enum.Enum
```

This enumeration lists all the available frame types used in any XBee protocol.

# Inherited properties:

```
name (String): the name (id) of this ApiFrameType. value (String): the value of this ApiFrameType.
```

## Values:

```
ApiFrameType.TX_64 = (0, 'TX (Transmit) Request 64-bit address')

ApiFrameType.TX_16 = (1, 'TX (Transmit) Request 16-bit address')

ApiFrameType.REMOTE_AT_COMMAND_REQUEST_WIFI = (7, 'Remote AT Command Request (Wi-Fi)')

ApiFrameType.AT_COMMAND = (8, 'AT Command')

ApiFrameType.AT_COMMAND_QUEUE = (9, 'AT Command Queue')

ApiFrameType.TRANSMIT_REQUEST = (16, 'Transmit Request')

ApiFrameType.EXPLICIT_ADDRESSING = (17, 'Explicit Addressing Command Frame')

ApiFrameType.REMOTE_AT_COMMAND_REQUEST = (23, 'Remote AT Command Request')

ApiFrameType.TX_SMS = (31, 'TX SMS')
```

```
ApiFrameType.TX IPV4 = (32, 'TX IPv4')
ApiFrameType.CREATE_SOURCE_ROUTE = (33, 'Create Source Route')
ApiFrameType.REGISTER JOINING DEVICE = (36, 'Register Joining Device')
ApiFrameType.SEND_DATA_REQUEST = (40, 'Send Data Request')
ApiFrameType.DEVICE_RESPONSE = (42, 'Device Response')
ApiFrameType.USER DATA RELAY REQUEST = (45, 'User Data Relay Request')
ApiFrameType.FILE SYSTEM REQUEST = (59, 'File System Request')
ApiFrameType.REMOTE_FILE_SYSTEM_REQUEST = (60, 'Remote File System Request')
ApiFrameType.SOCKET CREATE = (64, 'Socket Create')
ApiFrameType.SOCKET_OPTION_REQUEST = (65, 'Socket Option Request')
ApiFrameType.SOCKET CONNECT = (66, 'Socket Connect')
ApiFrameType.SOCKET CLOSE = (67, 'Socket Close')
ApiFrameType.SOCKET SEND = (68, 'Socket Send (Transmit)')
ApiFrameType.SOCKET_SENDTO = (69, 'Socket SendTo (Transmit Explicit Data): IPv4')
ApiFrameType.SOCKET BIND = (70, 'Socket Bind/Listen')
ApiFrameType.RX 64 = (128, 'RX (Receive) Packet 64-bit Address')
ApiFrameType.RX 16 = (129, 'RX (Receive) Packet 16-bit Address')
ApiFrameType.RX_IO_64 = (130, 'IO Data Sample RX 64-bit Address Indicator')
ApiFrameType.RX IO 16 = (131, 'IO Data Sample RX 16-bit Address Indicator')
ApiFrameType.REMOTE_AT_COMMAND_RESPONSE_WIFI = (135, 'Remote AT Command
Response (Wi-Fi)')
ApiFrameType.AT COMMAND RESPONSE = (136, 'AT Command Response')
ApiFrameType.TX_STATUS = (137, 'TX (Transmit) Status')
ApiFrameType.MODEM STATUS = (138, 'Modem Status')
ApiFrameType.TRANSMIT STATUS = (139, 'Transmit Status')
ApiFrameType.DIGIMESH ROUTE INFORMATION = (141, 'Route Information')
ApiFrameType.IO_DATA_SAMPLE_RX_INDICATOR_WIFI = (143, 'IO Data Sample RX
Indicator (Wi-Fi)')
ApiFrameType.RECEIVE PACKET = (144, 'Receive Packet')
ApiFrameType.EXPLICIT_RX_INDICATOR = (145, 'Explicit RX Indicator')
ApiFrameType.IO_DATA_SAMPLE_RX_INDICATOR = (146, 'IO Data Sample RX Indicator')
ApiFrameType.REMOTE AT COMMAND RESPONSE = (151, 'Remote Command Response')
ApiFrameType.RX SMS = (159, 'RX SMS')
ApiFrameType.OTA_FIRMWARE_UPDATE_STATUS = (160, 'OTA Firmware Update Status')
ApiFrameType.ROUTE RECORD INDICATOR = (161, 'Route Record Indicator')
ApiFrameType.REGISTER_JOINING_DEVICE_STATUS = (164, 'Register Joining Device Status')
ApiFrameType.USER_DATA_RELAY_OUTPUT = (173, 'User Data Relay Output')
ApiFrameType.RX IPV4 = (176, 'RX IPv4')
ApiFrameType.SEND DATA RESPONSE = (184, 'Send Data Response')
ApiFrameType.DEVICE_REQUEST = (185, 'Device Request')
ApiFrameType.DEVICE_RESPONSE_STATUS = (186, 'Device Response Status')
ApiFrameType.FILE SYSTEM RESPONSE = (187, 'File System Response')
ApiFrameType.REMOTE_FILE_SYSTEM_RESPONSE = (188, 'Remote File System Response')
ApiFrameType.SOCKET_CREATE_RESPONSE = (192, 'Socket Create Response')
ApiFrameType.SOCKET OPTION RESPONSE = (193, 'Socket Option Response')
ApiFrameType.SOCKET_CONNECT_RESPONSE = (194, 'Socket Connect Response')
ApiFrameType.SOCKET_CLOSE_RESPONSE = (195, 'Socket Close Response')
```

```
ApiFrameType.SOCKET_LISTEN_RESPONSE = (198, 'Socket Listen Response')
ApiFrameType.SOCKET_NEW_IPV4_CLIENT = (204, 'Socket New IPv4 Client')
ApiFrameType.SOCKET_RECEIVE = (205, 'Socket Receive')
ApiFrameType.SOCKET_RECEIVE_FROM = (206, 'Socket Receive From')
ApiFrameType.SOCKET_STATE = (207, 'Socket State')
ApiFrameType.FRAME_ERROR = (254, 'Frame Error')
ApiFrameType.GENERIC = (255, 'Generic')
ApiFrameType.UNKNOWN = (-1, 'Unknown Packet')
```

#### code

Returns the code of the ApiFrameType element.

**Returns** the code of the ApiFrameType element.

Return type Integer

# description

Returns the description of the ApiFrameType element.

**Returns** the description of the ApiFrameType element.

Return type Integer

# digi.xbee.packets.base module

```
class digi.xbee.packets.base.DictKeys
    Bases: enum.Enum
```

This enumeration contains all keys used in dictionaries returned by to\_dict() method of XBeePacket.

Bases: object

This abstract class represents the basic structure of an XBee packet. Derived classes should implement their own payload generation depending on their type.

Generic actions like checksum compute or packet length calculation is performed here.

Class constructor. Instantiates a new XBeePacket object.

```
Parameters op_mode (OperatingMode, optional, default='OperatingMode.API_MODE') – The mode in which the frame was captured.
```

## op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

## See also:

```
factory
```

## output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

Return type Dictionary

## static create\_packet (raw, operating\_mode)

Abstract method. Creates a full XBeePacket with the given parameters. This function ensures that the XBeePacket returned is valid and is well built (if not exceptions are raised).

If \_OPERATING\_MODE is API2 (API escaped) this method des-escape 'raw' and build the XBeePacket. Then, you can use XBeePacket.output() to get the escaped bytearray or not escaped.

#### **Parameters**

- raw (Bytearray) bytearray with which the frame will be built. Must be a full frame represented by a bytearray.
- **operating\_mode** (*OperatingMode*) The mode in which the frame ('byteArray') was captured.

**Returns** the XBee packet created.

Return type XBeePacket

**Raises** InvalidPacketException — if something is wrong with raw and the packet cannot be built well.

## get\_frame\_spec\_data()

Returns the data between the length field and the checksum field as bytearray. This data is never escaped.

#### Returns

the data between the length field and the checksum field as bytearray.

**Return type** Bytearray

See also:

factory

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

```
Returns data unescaped.
```

```
Return type Bytearray
```

This abstract class provides the basic structure of a API frame. Derived classes should implement their own methods to generate the API data and frame ID in case they support it.

Basic operations such as frame type retrieval are performed in this class.

## See also:

*XBeePacket* 

Class constructor. Instantiates a new XBeeAPIPacket object with the provided parameters.

## **Parameters**

- api\_frame\_type (ApiFrameType or Integer) The API frame type.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# See also:

```
ApiFrameType
XBeePacket

get_frame_spec_data()
   Override method.
   See also:

   XBeePacket.get_frame_spec_data()

get_frame_type()
   Returns the frame type of this packet.
        Return type ApiFrameType
   See also:

   ApiFrameType

get_frame_type_value()
   Returns the frame type integer value of this packet.
```

**Returns** the frame type integer value of this packet.

**Return type** Integer

See also:

ApiFrameType

## is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

# needs\_id()

Returns whether the packet requires frame ID or not.

**Returns** *True* if the packet needs frame ID, *False* otherwise.

Return type Boolean

## static create\_packet (raw, operating\_mode)

Abstract method. Creates a full XBeePacket with the given parameters. This function ensures that the XBeePacket returned is valid and is well built (if not exceptions are raised).

If \_OPERATING\_MODE is API2 (API escaped) this method des-escape 'raw' and build the XBeePacket. Then, you can use XBeePacket.output() to get the escaped bytearray or not escaped.

## **Parameters**

- raw (Bytearray) bytearray with which the frame will be built. Must be a full frame represented by a bytearray.
- **operating\_mode** (OperatingMode) The mode in which the frame ('byteArray') was captured.

Returns the XBee packet created.

Return type XBeePacket

**Raises** InvalidPacketException – if something is wrong with *raw* and the packet cannot be built well.

# get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

**Return type** Integer

See also:

factory

```
op mode
```

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

Return type Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a basic and Generic XBee packet.

See also:

*XBeeAPIPacket* 

Class constructor. Instantiates a GenericXBeePacket object with the provided parameters.

## **Parameters**

- data (bytearray) the frame specific data without frame type and frame ID.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

See also:

```
factory
XBeeAPIPacket
```

**static create\_packet** (raw, operating\_mode=<OperatingMode.API\_MODE: (1, 'API mode')>) Override method.

**Returns** the GenericXBeePacket generated.

## Return type GenericXBeePacket

#### Raises

- InvalidPacketException if the bytearray length is less than 5. (start delim. + length (2 bytes) + frame type + checksum = 5 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. GENERIC.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()
```

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

factory

# get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
```

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an unknown XBee packet.

See also:

XBeeAPIPacket

Class constructor. Instantiates a *UnknownXBeePacket* object with the provided parameters.

#### **Parameters**

- api\_frame (Integer) the API frame integer value of this packet.
- data (bytearray) the frame specific data without frame type and frame ID.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

See also:

```
factory
XBeeAPIPacket
```

static create\_packet (raw, operating\_mode=<OperatingMode.API\_MODE: (1, 'API mode')>)
 Override method.

**Returns** the UnknownXBeePacket generated.

Return type UnknownXBeePacket

#### Raises

- InvalidPacketException if the bytearray length is less than 5. (start delim. + length (2 bytes) + frame type + checksum = 5 bytes).
- InvalidPacketException if the length field of 'raw' is different its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidOperatingModeException if operating\_mode is not supported.

See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

```
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
```

# is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

## Return type Boolean

```
op_mode
```

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

# needs\_id()

Override method.

See also:

```
XBeeAPIPacket.needs_id()
```

# digi.xbee.packets.cellular module

This class represents an RX (Receive) SMS packet. Packet is built using the parameters of the constructor or providing a valid byte array.

See also:

```
TXSMSPacket
XBeeAPIPacket
```

Class constructor. Instantiates a new RXSMSPacket object with the provided parameters.

#### **Parameters**

- phone\_number (String) Phone number of the device that sent the SMS.
- data (String or bytearray) Packet data (text of the SMS).
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError if length of *phone\_number* is greater than 20.
- ValueError if *phone\_number* is not a valid phone number.

## static create\_packet (raw, operating\_mode)

Override method.

Returns RXSMSPacket

#### Raises

- InvalidPacketException if the bytearray length is less than 25. (start delim + length (2 bytes) + frame type + phone number (20 bytes) + checksum = 25 bytes)
- InvalidPacketException if the length field of *raw* is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of *raw* is not the header byte. See SPECIAL BYTE.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different than  ${\tt ApiFrameType.}$  RX\_SMS.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
needs_id()
Override method.
```

## See also:

```
XBeeAPIPacket.needs_id()
```

## get\_phone\_number\_byte\_array()

Returns the phone number byte array.

**Returns** phone number of the device that sent the SMS.

Return type Bytearray

```
phone_number
     Returns the phone number of the device that sent the SMS.
         Returns phone number of the device that sent the SMS.
         Return type String
data
     Returns the data of the packet (SMS text).
         Returns the data of the packet.
         Return type String
frame_id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
```

**Returns** the frame type integer value of this packet.

Returns the frame type integer value of this packet.

Return type Integer

*ApiFrameType* 

get\_frame\_type\_value()

## See also:

```
ApiFrameType
```

## is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

## op mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

#### output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

**Return type** Bytearray

Dascs. digi.xbee.packets.base.xbeexiii acket

This class represents a TX (Transmit) SMS packet. Packet is built using the parameters of the constructor or providing a valid byte array.

See also:

```
RXSMSPacket
XBeeAPIPacket
```

Class constructor. Instantiates a new TXSMSPacket object with the provided parameters.

## **Parameters**

- **frame\_id** (*Integer*) the frame ID. Must be between 0 and 255.
- phone\_number (String) the phone number.
- data (String or bytearray) this packet's data.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError if *frame\_id* is not between 0 and 255.
- ValueError if length of *phone\_number* is greater than 20.
- ValueError if *phone\_number* is not a valid phone number.

## See also:

XBeeAPIPacket

## static create\_packet (raw, operating\_mode)

Override method.

Returns TXSMSPacket

#### Raises

- InvalidPacketException if the bytearray length is less than 27. (start delim, length (2 bytes), frame type, frame id, transmit options, phone number (20 bytes), checksum)
- InvalidPacketException if the length field of *raw* is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of *raw* is not the header byte. See SPECIAL\_BYTE.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different than ApiFrameType. TX SMS.
- InvalidOperatingModeException if operating\_mode is not supported.

# See also:

```
XBeePacket.create_packet()
needs_id()
Override method.
See also:
```

XBeeAPIPacket.needs\_id()

```
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
get_phone_number_byte_array()
    Returns the phone number byte array.
```

Returns phone number of the device that sent the SMS.

## **Return type** Bytearray

### is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

**Return type** Boolean

## op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

**Return type** Bytearray

## to dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

## phone\_number

Returns the phone number of the transmitter device.

**Returns** the phone number of the transmitter device.

Return type String

#### data

Returns the data of the packet (SMS text).

**Returns** packet's data.

**Return type** Bytearray

## digi.xbee.packets.common module

This class represents an AT command packet.

Used to query or set module parameters on the local device. This API command applies changes after executing the command. (Changes made to module parameters take effect once changes are applied.).

Command response is received as an ATCommResponsePacket.

#### See also:

```
ATCommResponsePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new ATCommPacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- command (String or bytearray) AT command of the packet.
- parameter (Bytearray, optional) the AT command parameter.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if length of *command* is different from 2.

### See also:

XBeeAPIPacket

## static create packet(raw, operating mode)

Override method.

Returns ATCommPacket

## Raises

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + command (2 bytes) + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. AT\_COMMAND.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

```
needs id()
     Override method.
     See also:
     XBeeAPIPacket.needs id()
command
     Returns the AT command of the packet.
         Returns the AT command of the packet.
         Return type String
parameter
     Returns the parameter of the packet.
         Returns the parameter of the packet.
         Return type Bytearray
frame_id
     Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
     Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
```

```
ApiFrameType
     get_frame_type_value()
          Returns the frame type integer value of this packet.
              Returns the frame type integer value of this packet.
              Return type Integer
          See also:
          ApiFrameType
     is_broadcast()
          Returns whether this packet is broadcast or not.
              Returns True if this packet is broadcast, False otherwise.
              Return type Boolean
     op_mode
          Retrieves the operating mode in which this packet was read.
              Returns The operating mode.
              Return type OperatingMode
     output (escaped=False)
          Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
              Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
              Returns raw bytearray of the XBeePacket.
              Return type Bytearray
     to_dict()
          Returns a dictionary with all information of the XBeePacket fields.
              Returns dictionary with all info of the XBeePacket fields.
              Return type Dictionary
     static unescape_data(data)
          Un-escapes the provided bytearray data.
              Parameters data (Bytearray) – the bytearray to unescape.
              Returns data unescaped.
              Return type Bytearray
class digi.xbee.packets.common.ATCommQueuePacket (frame_id,
                                                                                           command,
                                                                  parameter=None,
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an AT command Queue packet.

Used to query or set module parameters on the local device.

op\_mode=<OperatingMode.API\_MODE:

(1, 'API mode')>)

In contrast to the ATCommPacket API packet, new parameter values are queued and not applied until either an ATCommPacket is sent or the applyChanges() method of the XBeeDevice class is issued.

Command response is received as an ATCommResponsePacket.

#### See also:

ATCommResponsePacket XBeeAPIPacket

Class constructor. Instantiates a new ATCommQueuePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- command (String or bytearray) the AT command of the packet.
- parameter (Bytearray, optional) the AT command parameter. Optional.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if length of command is different from 2.

## See also:

*XBeeAPIPacket* 

# static create\_packet (raw, operating\_mode)

Override method.

Returns ATCommQueuePacket

## Raises

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + command + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. AT\_COMMAND\_QUEUE.
- $\bullet \ \, \hbox{InvalidOperatingModeException-if} \ \textit{operating\_mode} \ \text{is not supported}.$

## See also:

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs_id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
command
    Returns the AT command of the packet.
        Returns the AT command of the packet.
        Return type String
parameter
    Returns the parameter of the packet.
        Returns the parameter of the packet.
        Return type Bytearray
frame id
    Returns the frame ID of the packet.
        Returns the frame ID of the packet.
        Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
        Returns checksum value of this XBeePacket.
        Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
        Returns the frame type of this packet.
```

```
Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
```

```
 \textbf{class} \  \, \textbf{digi.xbee.packets.common.ATCommResponsePacket} \, \, (\textit{frame\_id}, \quad \textit{command}, \quad \textit{response\_status} = < ATCommandStatus.OK: \\ (0, \quad `Status \quad OK') >, \\ comm\_value = None, \\ op\_mode = < OperatingMode.API\_MODE: \\ (1, `API mode') >)
```

 $Bases: \ \textit{digi.xbee.packets.base.XBeeAPIPacket}$ 

This class represents an AT command response packet.

In response to an AT command message, the module will send an AT command response message. Some commands will send back multiple frames (for example, the *ND* - Node Discover command).

This packet is received in response of an ATCommPacket.

Response also includes an ATCommandStatus object with the status of the AT command.

#### See also:

ATCommPacket
ATCommandStatus
XBeeAPIPacket

Class constructor. Instantiates a new ATCommResponsePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (Integer) the frame ID of the packet. Must be between 0 and 255.
- command (String or bytearray) the AT command of the packet.
- response\_status (ATCommandStatus or Integer) the status of the AT command.
- comm\_value (Bytearray, optional) the AT command response value.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if length of *command* is different from 2.

## See also:

ATCommandStatus XBeeAPIPacket

static create\_packet(raw, operating\_mode)

Override method.

Returns ATCommResponsePacket

## Raises

• InvalidPacketException – if the bytearray length is less than 9. (start delim. + length (2 bytes) + frame type + frame id + at command (2 bytes) + command status + checksum = 9 bytes).

- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. AT\_COMMAND\_RESPONSE.
- InvalidPacketException if the command status field is not a valid value. See ATCommandStatus.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

## See also:

```
XBeeAPIPacket.needs id()
```

## command

Returns the AT command of the packet.

**Returns** the AT command of the packet.

Return type String

## command\_value

Returns the AT command response value.

**Returns** the AT command response value.

**Return type** Bytearray

# real\_status

Returns the AT command response status of the packet.

**Returns** the AT command response status of the packet.

Return type Integer

## status

Returns the AT command response status of the packet.

**Returns** the AT command response status of the packet.

Return type ATCommandStatus

See also:

```
ATCommandStatus
```

```
frame_id
```

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

factory

```
get_frame_spec_data()
```

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
```

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

```
ApiFrameType
```

## get\_frame\_type\_value()

Returns the frame type integer value of this packet.

**Returns** the frame type integer value of this packet.

Return type Integer

See also:

*ApiFrameType* 

### is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

# op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

```
class digi.xbee.packets.common.ReceivePacket (x64bit\_addr, x16bit\_addr, rx\_options, rf\_data=None, op\_mode=<OperatingMode.API\_MODE: (1, 'API mode')>)
Bases: digi.xbee.packets.base.XBeeAPIPacket
```

This class represents a receive packet. Packet is built using the parameters of the constructor or providing a valid byte array.

When the module receives an RF packet, it is sent out the UART using this message type.

This packet is received when external devices send transmit request packets to this module.

Among received data, some options can also be received indicating transmission parameters.

See also:

```
TransmitPacket
ReceiveOptions
XBeeAPIPacket
```

Class constructor. Instantiates a new ReceivePacket object with the provided parameters.

# **Parameters**

- x64bit\_addr (XBee64BitAddress) the 64-bit source address.
- x16bit\_addr (XBee16BitAddress) the 16-bit source address.
- rx\_options (Integer) bitfield indicating the receive options.
- rf\_data (Bytearray, optional) received RF data.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### See also:

ReceiveOptions XBee16BitAddress XBee64BitAddress XBeeAPIPacket

## static create\_packet (raw, operating\_mode)

Override method.

Returns ATCommResponsePacket

## Raises

- InvalidPacketException if the bytearray length is less than 16. (start delim. + length (2 bytes) + frame type + 64bit addr. + 16bit addr. + Receive options + checksum = 16 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. RECEIVE\_PACKET.
- $\bullet \ {\tt InvalidOperatingModeException-if} \ operating\_mode \ is \ not \ supported.$

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()
```

# is broadcast() Override method. See also: XBeeAPIPacket.is broadcast() x64bit\_source\_addr Returns the 64-bit source address. **Returns** the 64-bit source address. Return type XBee64BitAddress See also: XBee64BitAddress x16bit\_source\_addr Returns the 16-bit source address. **Returns** the 16-bit source address. Return type XBee16BitAddress See also: XBee16BitAddress receive\_options Returns the receive options bitfield. **Returns** the receive options bitfield. **Return type** Integer See also: ReceiveOptions rf\_data Returns the received RF data.

Returns the received RF data.

Return type Bytearray

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
```

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

```
to_dict()
```

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

```
static unescape_data(data)
```

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

```
class digi.xbee.packets.common.RemoteATCommandPacket (frame\_id, x64bit\_addr, x16bit\_addr, tx\_options, command, parameter=None, op\_mode=<OperatingMode.API\_MODE: (1, 'API mode')>)
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Remote AT command Request packet. Packet is built using the parameters of the constructor or providing a valid byte array.

Used to query or set module parameters on a remote device. For parameter changes on the remote device to take effect, changes must be applied, either by setting the apply changes options bit, or by sending an AC command to the remote node.

Remote command options are set as a bitfield.

 $If configured, command \ response \ is \ received \ as \ a \ \textit{RemoteATCommandResponsePacket}.$ 

See also:

```
RemoteATCommandResponsePacket XBeeAPIPacket
```

Class constructor. Instantiates a new RemoteATCommandPacket object with the provided parameters.

# **Parameters**

- **frame\_id** (*integer*) the frame ID of the packet.
- x64bit addr (XBee64BitAddress) the 64-bit destination address.
- x16bit\_addr (XBee16BitAddress) the 16-bit destination address.
- **tx\_options** (Integer) bitfield of supported transmission options.
- command (String or bytearray) AT command to send.
- parameter (Bytearray, optional) AT command parameter.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** 

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if length of *command* is different from 2.

# See also:

RemoteATCmdOptions XBee16BitAddress XBee64BitAddress XBeeAPIPacket

# static create\_packet (raw, operating\_mode)

Override method.

Returns RemoteATCommandPacket

## Raises

- InvalidPacketException if the Bytearray length is less than 19. (start delim. + length (2 bytes) + frame type + frame id + 64bit addr. + 16bit addr. + transmit options + command (2 bytes) + checksum = 19 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. REMOTE\_AT\_COMMAND\_REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

# See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

# needs\_id()

Override method.

## See also:

```
XBeeAPIPacket.needs_id()
```

## x64bit dest addr

Returns the 64-bit destination address.

**Returns** the 64-bit destination address.

Return type XBee64BitAddress

## See also:

XBee64BitAddress

# x16bit\_dest\_addr

Returns the 16-bit destination address.

**Returns** the 16-bit destination address.

Return type XBee16BitAddress

See also:

XBee16BitAddress

# transmit\_options

Returns the transmit options bitfield.

**Returns** the transmit options bitfield.

Return type Integer

See also:

RemoteATCmdOptions

# parameter

Returns the AT command parameter.

Returns the AT command parameter.

Return type Bytearray

# command

Returns the AT command.

Returns the AT command.

Return type String

## frame id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

## get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
    Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
    Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
```

```
to dict()
```

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

#### static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

```
class digi.xbee.packets.common.RemoteATCommandResponsePacket(frame_id,
```

```
x64bit_addr,
x16bit_addr, com-
mand, resp_status,
comm_value=None,
op_mode=<OperatingMode.API_MODE:
(1, 'API mode')>)
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a remote AT command response packet. Packet is built using the parameters of the constructor or providing a valid byte array.

If a module receives a remote command response RF data frame in response to a remote AT command request, the module will send a remote AT command response message out the UART. Some commands may send back multiple frames, for example, Node Discover (*ND*) command.

This packet is received in response of a RemoteATCommandPacket.

Response also includes an object with the status of the AT command.

## See also:

RemoteATCommandPacket
ATCommandStatus
XBeeAPIPacket

Class constructor. Instantiates a new RemoteATCommandResponsePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- x64bit\_addr (XBee64BitAddress) the 64-bit source address
- x16bit\_addr (XBee16BitAddress) the 16-bit source address.
- command (String or bytearray) the AT command of the packet.
- resp\_status (ATCommandStatus or Integer) the status of the AT command.
- comm\_value (Bytearray, optional) the AT command response value. Optional.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if length of command is different from 2.

## See also:

ATCommandStatus XBee16BitAddress XBee64BitAddress XBeeAPIPacket

# static create\_packet (raw, operating\_mode)

Override method.

**Returns** RemoteATCommandResponsePacket.

## Raises

- InvalidPacketException if the bytearray length is less than 19. (start delim. + length (2 bytes) + frame type + frame id + 64bit addr. + 16bit addr. + receive options + command (2 bytes) + checksum = 19 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. REMOTE\_AT\_COMMAND\_RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

# See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

See also:

```
XBeeAPIPacket.needs_id()
```

#### command

Returns the AT command of the packet.

**Returns** the AT command of the packet.

# Return type String

# command\_value

Returns the AT command response value.

**Returns** the AT command response value.

**Return type** Bytearray

## real\_status

Returns the AT command response status of the packet.

**Returns** the AT command response status of the packet.

Return type Integer

#### status

Returns the AT command response status of the packet.

**Returns** the AT command response status of the packet.

Return type ATCommandStatus

See also:

ATCommandStatus

# x64bit\_source\_addr

Returns the 64-bit source address.

Returns the 64-bit source address.

Return type XBee64BitAddress

See also:

XBee64BitAddress

# x16bit\_source\_addr

Returns the 16-bit source address.

**Returns** the 16-bit source address.

Return type XBee16BitAddress

See also:

XBee16BitAddress

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
```

**Returns** The operating mode.

# Return type OperatingMode

```
output (escaped=False)
```

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

```
class digi.xbee.packets.common.TransmitPacket (frame\_id, x64bit\_addr, s16bit\_addr, s16bi
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a transmit request packet. Packet is built using the parameters of the constructor or providing a valid API byte array.

A transmit request API frame causes the module to send data as an RF packet to the specified destination.

The 64-bit destination address should be set to 0x00000000000FFFF for a broadcast transmission (to all devices).

For all other transmissions, setting the 16-bit address to the correct 16-bit address can help improve performance when transmitting to multiple destinations.

If a 16-bit address is not known, this field should be set to 0xFFFE (unknown).

The transmit status frame (ApiFrameType.TRANSMIT\_STATUS) will indicate the discovered 16-bit address, if successful (see *TransmitStatusPacket*).

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

The maximum number of payload bytes can be read with the NP command.

Several transmit options can be set using the transmit options bitfield.

See also:

TransmitOptions

```
XBee16BitAddress.COORDINATOR_ADDRESS
XBee16BitAddress.UNKNOWN_ADDRESS
XBee64BitAddress.BROADCAST_ADDRESS
XBee64BitAddress.COORDINATOR_ADDRESS
XBeeAPIPacket
```

Class constructor. Instantiates a new TransmitPacket object with the provided parameters.

## **Parameters**

- **frame\_id** (*integer*) the frame ID of the packet.
- x64bit addr (XBee64BitAddress) the 64-bit destination address.
- x16bit\_addr (XBee16BitAddress) the 16-bit destination address.
- **broadcast\_radius** (Integer) maximum number of hops a broadcast transmission can occur.
- tx\_options (Integer) bitfield of supported transmission options.
- rf\_data (Bytearray, optional) RF data that is sent to the destination device.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### See also:

TransmitOptions XBee16BitAddress XBee64BitAddress XBeeAPIPacket

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

# static create\_packet (raw, operating\_mode)

Override method.

Returns TransmitPacket.

## Raises

- InvalidPacketException if the bytearray length is less than 18. (start delim. + length (2 bytes) + frame type + frame id + 64bit addr. + 16bit addr. + broadcast radius + Transmit options + checksum = 18 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. TRANSMIT\_REQUEST.
- $\bullet \ \, {\tt InvalidOperatingModeException-if} \ operating\_mode \ is \ not \ supported.$

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()
```

## rf\_data

Returns the RF data to send.

**Returns** the RF data to send.

Return type Bytearray

## transmit\_options

Returns the transmit options bitfield.

**Returns** the transmit options bitfield.

Return type Integer

See also:

TransmitOptions

# broadcast\_radius

Returns the broadcast radius. Broadcast radius is the maximum number of hops a broadcast transmission.

Returns the broadcast radius.

Return type Integer

# x64bit\_dest\_addr

Returns the 64-bit destination address.

**Returns** the 64-bit destination address.

Return type XBee64BitAddress

See also:

XBee64BitAddress

# x16bit\_dest\_addr

Returns the 16-bit destination address.

**Returns** the 16-bit destination address.

```
Return type XBee16BitAddress
    See also:
     XBee16BitAddress
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
```

## ApiFrameType

# is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

**Return type** Boolean

# op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a transmit status packet. Packet is built using the parameters of the constructor or providing a valid raw byte array.

When a Transmit Request is completed, the module sends a transmit status message. This message will indicate if the packet was transmitted successfully or if there was a failure.

This packet is the response to standard and explicit transmit requests.

See also:

TransmitPacket

Class constructor. Instantiates a new TransmitStatusPacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- x16bit\_addr (XBee16BitAddress) 16-bit network address the packet was delivered to
- tx\_retry\_count (Integer) the number of application transmission retries that took place.
- transmit\_status (TransmitStatus, optional) transmit status. Default: SUC-CESS.
- discovery\_status (DiscoveryStatus, optional) discovery status. Default: NO\_DISCOVERY\_OVERHEAD.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

#### See also:

DiscoveryStatus TransmitStatus XBee16BitAddress XBeeAPIPacket

# static create\_packet(raw, operating\_mode)

Override method.

Returns TransmitStatusPacket

# Raises

- InvalidPacketException if the bytearray length is less than 11. (start delim. + length (2 bytes) + frame type + frame id + 16bit addr. + transmit retry count + delivery status + discovery status + checksum = 11 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. TRANSMIT STATUS.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

# needs\_id()

Override method.

See also:

XBeeAPIPacket.needs\_id()

# x16bit\_dest\_addr

Returns the 16-bit destination address.

**Returns** the 16-bit destination address.

Return type XBee16BitAddress

See also:

XBee16BitAddress

# transmit\_status

Returns the transmit status.

**Returns** the transmit status.

Return type TransmitStatus

See also:

TransmitStatus

# ${\tt transmit\_retry\_count}$

Returns the transmit retry count.

**Returns** the transmit retry count.

**Return type** Integer

# discovery\_status

Returns the discovery status.

**Returns** the discovery status.

Return type DiscoveryStatus

See also:

DiscoveryStatus

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
```

**Returns** The operating mode.

#### Return type OperatingMode

```
output (escaped=False)
```

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

Return type Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

class digi.xbee.packets.common.ModemStatusPacket (modem\_status,

op\_mode=<OperatingMode.API\_MODE:
(1, 'API mode')>)

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a modem status packet. Packet is built using the parameters of the constructor or providing a valid API raw byte array.

RF module status messages are sent from the module in response to specific conditions and indicates the state of the modem in that moment.

#### See also:

```
XBeeAPIPacket
```

Class constructor. Instantiates a new ModemStatusPacket object with the provided parameters.

#### **Parameters**

- modem\_status (ModemStatus) the modem status event.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# See also:

```
ModemStatus
XBeeAPIPacket
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns ModemStatusPacket.

#### Raises

- InvalidPacketException if the bytearray length is less than 6. (start delim. + length (2 bytes) + frame type + modem status + checksum = 6 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. MODEM\_STATUS.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

#### needs\_id()

Override method.

See also:

```
XBeeAPIPacket.needs_id()
```

### modem\_status

Returns the modem status event.

**Returns** The modem status event.

Return type ModemStatus

See also:

ModemStatus

#### frame id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

# get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

```
See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
     ApiFrameType
is_broadcast()
    Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
    Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
```

### **Return type** Bytearray

```
to dict()
```

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

```
static unescape_data(data)
```

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an IO data sample RX indicator packet. Packet is built using the parameters of the constructor or providing a valid API byte array.

When the module receives an IO sample frame from a remote device, it sends the sample out the UART using this frame type (when AO=0). Only modules running API firmware will send IO samples out the UART.

Among received data, some options can also be received indicating transmission parameters.

### See also:

```
XBeeAPIPacket
ReceiveOptions
```

Class constructor. Instantiates a new IODataSampleRxIndicatorPacket object with the provided parameters.

#### **Parameters**

- x64bit\_addr (XBee64BitAddress) the 64-bit source address.
- x16bit\_addr (XBee16BitAddress) the 16-bit source address.
- **rx\_options** (*Integer*) bitfield indicating the receive options.
- rf\_data (Bytearray, optional) received RF data.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *rf\_data* is not *None* and it's not valid for create an *IOSample*.

#### See also:

```
IOSample
ReceiveOptions
```

XBee16BitAddress XBee64BitAddress XBeeAPIPacket

### static create\_packet (raw, operating\_mode)

Override method.

Returns IODataSampleRxIndicatorPacket.

#### Raises

- InvalidPacketException if the bytearray length is less than 20. (start delim. + length (2 bytes) + frame type + 64bit addr. + 16bit addr. + rf data (5 bytes) + checksum = 20 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType.
   IO\_DATA\_SAMPLE\_RX\_INDICATOR.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

is_broadcast()
Override method.
See also:

XBeeAPIPacket.is_broadcast()
```

# x64bit\_source\_addr

Returns the 64-bit source address.

**Returns** the 64-bit source address.

```
Return type XBee64BitAddress
    See also:
     XBee64BitAddress
x16bit source addr
    Returns the 16-bit source address.
         Returns the 16-bit source address.
         Return type XBee16BitAddress
    See also:
     XBee16BitAddress
receive_options
    Returns the receive options bitfield.
         Returns the receive options bitfield.
         Return type Integer
    See also:
     ReceiveOptions
rf data
    Returns the received RF data.
         Returns the received RF data.
         Return type Bytearray
io sample
    Returns the IO sample corresponding to the data contained in the packet.
         Returns
            the IO sample of the packet, None if the packet has not any data or if the sample could not
               be generated correctly.
         Return type IOSample
    See also:
     IOSample
```

# $frame_id$

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

```
Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
     ApiFrameType
op_mode
    Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
```

2.6. API reference 289

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

output (escaped=False)

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an explicit addressing command packet. Packet is built using the parameters of the constructor or providing a valid API payload.

Allows application layer fields (endpoint and cluster ID) to be specified for a data transmission. Similar to the transmit request, but also requires application layer addressing fields to be specified (endpoints, cluster ID, profile ID). An explicit addressing request API frame causes the module to send data as an RF packet to the specified destination, using the specified source and destination endpoints, cluster ID, and profile ID.

The 64-bit destination address should be set to 0x00000000000FFF for a broadcast transmission (to all devices).

For all other transmissions, setting the 16-bit address to the right 16-bit address can help improve performance when transmitting to multiple destinations.

If a 16-bit address is not known, this field should be set to 0xFFFE (unknown).

The transmit status frame (ApiFrameType.TRANSMIT\_STATUS) will indicate the discovered 16-bit address, if successful (see *TransmitStatusPacket*)).

The broadcast radius can be set from  $\theta$  up to NH. If set to  $\theta$ , the value of NH specifies the broadcast radius (recommended). This parameter is only used for broadcast transmissions.

The maximum number of payload bytes can be read with the *NP* command. Note: if source routing is used, the RF payload will be reduced by two bytes per intermediate hop in the source route.

Several transmit options can be set using the transmit options bitfield.

See also:

```
TransmitOptions
XBee16BitAddress.COORDINATOR_ADDRESS
XBee16BitAddress.UNKNOWN_ADDRESS
XBee64BitAddress.BROADCAST_ADDRESS
XBee64BitAddress.COORDINATOR_ADDRESS
ExplicitRXIndicatorPacket
XBeeAPIPacket
```

Class constructor. . Instantiates a new ExplicitAddressingPacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- x64bit addr (XBee64BitAddress) the 64-bit address.
- x16bit\_addr (XBee16BitAddress) the 16-bit address.
- **src\_endpoint** (*Integer*) source endpoint. 1 byte.
- **dest\_endpoint** (*Integer*) destination endpoint. 1 byte.
- **cluster\_id** (*Integer*) cluster id. Must be between 0 and 0xFFFF.
- profile\_id (Integer) profile id. Must be between 0 and 0xFFFF.
- **broadcast\_radius** (Integer) maximum number of hops a broadcast transmission can occur.
- transmit\_options (Integer) bitfield of supported transmission options.
- $rf_{data}$  (Bytearray, optional) RF data that is sent to the destination device.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *frame\_id*, *src\_endpoint* or *dst\_endpoint* are less than 0 or greater than 255.
- ValueError if lengths of *cluster\_id* or *profile\_id* (respectively) are less than 0 or greater than 0xFFFF.

### See also:

```
XBee16BitAddress
XBee64BitAddress
TransmitOptions
XBeeAPIPacket
```

```
static create_packet (raw, operating_mode)
```

Override method.

Returns ExplicitAddressingPacket.

Raises

- InvalidPacketException if the bytearray length is less than 24. (start delim. + length (2 bytes) + frame type + frame ID + 64bit addr. + 16bit addr. + source endpoint + dest. endpoint + cluster ID (2 bytes) + profile ID (2 bytes) + broadcast radius + transmit options + checksum = 24 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. EXPLICIT\_ADDRESSING.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

#### frame id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

# get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

### get\_frame\_type()

Returns the frame type of this packet.

**Returns** the frame type of this packet.

```
Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
     ApiFrameType
is broadcast()
    Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
needs id()
    Override method.
     See also:
     XBeeAPIPacket.needs_id()
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
```

```
Parameters data (Bytearray) – the bytearray to unescape.
```

Returns data unescaped.

Return type Bytearray

### source\_endpoint

Returns the source endpoint of the transmission.

**Returns** the source endpoint of the transmission.

Return type Integer

# dest\_endpoint

Returns the destination endpoint of the transmission.

**Returns** the destination endpoint of the transmission.

Return type Integer

### cluster\_id

Returns the cluster ID of the transmission.

**Returns** the cluster ID of the transmission.

Return type Integer

### profile\_id

Returns the profile ID of the transmission.

**Returns** Integer: the profile ID of the transmission.

#### rf data

Returns the RF data to send.

Returns the RF data to send.

Return type Bytearray

# transmit\_options

Returns the transmit options bitfield.

**Returns** the transmit options bitfield.

Return type Integer

See also:

TransmitOptions

### broadcast\_radius

Returns the broadcast radius. Broadcast radius is the maximum number of hops a broadcast transmission.

**Returns** the broadcast radius.

Return type Integer

#### x64bit dest addr

Returns the 64-bit destination address.

**Returns** the 64-bit destination address.

Return type XBee64BitAddress

See also:

XBee64BitAddress

# x16bit\_dest\_addr

Returns the 16-bit destination address.

**Returns** the 16-bit destination address.

Return type XBee16BitAddress

See also:

XBee16BitAddress

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an explicit RX indicator packet. Packet is built using the parameters of the constructor or providing a valid API payload.

When the modern receives an RF packet it is sent out the UART using this message type (when AO=1).

This packet is received when external devices send explicit addressing packets to this module.

Among received data, some options can also be received indicating transmission parameters.

See also:

ReceiveOptions ExplicitAddressingPacket XBeeAPIPacket

Class constructor. Instantiates a new ExplicitRXIndicatorPacket object with the provided parameters.

#### **Parameters**

- x64bit\_addr (XBee64BitAddress) the 64-bit source address.
- x16bit\_addr (XBee16BitAddress) the 16-bit source address.
- **src\_endpoint** (*Integer*) source endpoint. 1 byte.
- **dest\_endpoint** (*Integer*) destination endpoint. 1 byte.
- cluster\_id (Integer) cluster ID. Must be between 0 and 0xFFFF.
- **profile\_id** (*Integer*) **profile** ID. Must be between 0 and 0xFFFF.
- **rx\_options** (*Integer*) bitfield indicating the receive options.

- rf\_data(Bytearray, optional) received RF data.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### **Raises**

- ValueError if *src\_endpoint* or *dst\_endpoint* are less than 0 or greater than 255.
- ValueError if lengths of *cluster\_id* or *profile\_id* (respectively) are different from 2.

#### See also:

```
XBee16BitAddress
XBee64BitAddress
ReceiveOptions
XBeeAPIPacket
```

#### frame id

Returns the frame ID of the packet.

Returns the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

```
get_frame_spec_data()
```

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
```

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

```
ApiFrameType
```

### get\_frame\_type\_value()

Returns the frame type integer value of this packet.

**Returns** the frame type integer value of this packet.

Return type Integer

See also:

ApiFrameType

### op\_mode

Retrieves the operating mode in which this packet was read.

Returns The operating mode.

Return type OperatingMode

output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

#### to dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

# static create\_packet (raw, operating\_mode)

Override method.

Returns ExplicitRXIndicatorPacket.

### Raises

- InvalidPacketException if the bytearray length is less than 22. (start delim. + length (2 bytes) + frame type + 64bit addr. + 16bit addr. + source endpoint + dest. endpoint + cluster ID (2 bytes) + profile ID (2 bytes) + receive options + checksum = 22 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).

- InvalidPacketException if the frame type is different from ApiFrameType. EXPLICIT\_RX\_INDICATOR.
- InvalidOperatingModeException if operating\_mode is not supported.

See also:

source\_endpoint

298

Returns the source endpoint of the transmission.

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs_id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
is_broadcast()
    Override method.
    See also:
    XBeeAPIPacket.is_broadcast()
x64bit_source_addr
    Returns the 64-bit source address.
        Returns the 64-bit source address.
        Return type XBee64BitAddress
    See also:
    XBee64BitAddress
x16bit_source_addr
    Returns the 16-bit source address.
        Returns the 16-bit source address.
        Return type XBee16BitAddress
    See also:
    XBee16BitAddress
```

Chapter 2. Contents

**Returns** the source endpoint of the transmission.

**Return type** Integer

# dest\_endpoint

Returns the destination endpoint of the transmission.

**Returns** the destination endpoint of the transmission.

**Return type** Integer

### cluster id

Returns the cluster ID of the transmission.

**Returns** the cluster ID of the transmission.

Return type Integer

### profile\_id

Returns the profile ID of the transmission.

**Returns** Integer: the profile ID of the transmission.

### receive\_options

Returns the receive options bitfield.

**Returns** the receive options bitfield.

Return type Integer

See also:

ReceiveOptions

#### rf data

Returns the received RF data.

Returns the received RF data.

**Return type** Bytearray

# digi.xbee.packets.devicecloud module

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a device request packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame type is sent out the serial port when the XBee module receives a valid device request from Device Cloud.

See also:

DeviceResponsePacket XBeeAPIPacket

Class constructor. Instantiates a new DeviceRequestPacket object with the provided parameters.

#### **Parameters**

- request\_id (Integer) number that identifies the device request. (0 has no special meaning)
- target (String) device request target.
- request\_data (Bytearray, optional) data of the request.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### Raises

- ValueError if *request\_id* is less than 0 or greater than 255.
- ValueError if length of *target* is greater than 255.

#### See also:

*XBeeAPIPacket* 

### static create\_packet (raw, operating\_mode)

Override method.

Returns DeviceRequestPacket

#### Raises

- InvalidPacketException if the bytearray length is less than 9. (start delim. + length (2 bytes) + frame type + request id + transport + flags + target length + checksum = 9 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. DEVICE\_REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

```
XBeeAPIPacket.needs id()
```

# request\_id

Returns the request ID of the packet.

**Returns** the request ID of the packet.

Return type Integer

# transport

Returns the transport of the packet.

**Returns** the transport of the packet.

Return type Integer

### flags

Returns the flags of the packet.

**Returns** the flags of the packet.

Return type Integer

### target

Returns the device request target of the packet.

**Returns** the device request target of the packet.

Return type String

# request\_data

Returns the data of the device request.

**Returns** the data of the device request.

Return type Bytearray

#### frame id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

### get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

**Returns** checksum value of this XBeePacket.

Return type Integer

See also:

factory

# get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
```

302

Returns data unescaped.

### Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a device response packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame type is sent to the serial port by the host in response to the <code>DeviceRequestPacket</code>. It should be sent within five seconds to avoid a timeout error.

#### See also:

DeviceRequestPacket XBeeAPIPacket

Class constructor. Instantiates a new <code>DeviceResponsePacket</code> object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- request\_id (Integer) device Request ID. This number should match the device request ID in the device request. Otherwise, an error will occur. (0 has no special meaning)
- response\_data (Bytearray, optional) data of the response.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *request\_id* is less than 0 or greater than 255.

#### See also:

*XBeeAPIPacket* 

static create\_packet (raw, operating\_mode)

Override method.

Returns DeviceResponsePacket

#### Raises

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + request id + reserved + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes  $2\ and\ 3$ )

- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. DEVICE RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs id()
```

### request\_id

Returns the request ID of the packet.

**Returns** the request ID of the packet.

Return type Integer

### request\_data

Returns the data of the device response.

**Returns** the data of the device response.

Return type Bytearray

### frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

## get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

**Returns** checksum value of this XBeePacket.

**Return type** Integer

See also:

factory

```
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get frame spec data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
```

### **Return type** Dictionary

### static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

class digi.xbee.packets.devicecloud.DeviceResponseStatusPacket (frame\_id, status,

```
op_mode=<OperatingMode.API_MODE
(1, 'API
mode')>)
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a device response status packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame type is sent to the serial port after the serial port sends a <code>DeviceResponsePacket</code>.

#### See also:

```
DeviceResponsePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new <code>DeviceResponseStatusPacket</code> object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **status** (DeviceCloudStatus) device response status.

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

### See also:

```
DeviceCloudStatus
XBeeAPIPacket
```

### static create\_packet (raw, operating\_mode)

Override method.

**Returns** DeviceResponseStatusPacket

### Raises

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + frame id + device response status + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.

- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. DEVICE\_RESPONSE\_STATUS.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()
```

#### status

Returns the status of the device response packet.

**Returns** the status of the device response packet.

 $\textbf{Return type} \ \textit{DeviceCloudStatus}$ 

See also:

*DeviceCloudStatus* 

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

### get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

factory

```
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get frame spec data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
```

Returns dictionary with all info of the XBeePacket fields.

### **Return type** Dictionary

#### static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

class digi.xbee.packets.devicecloud.FrameErrorPacket (frame\_error,

op\_mode=<OperatingMode.API\_MODE:
(1, 'API mode')>)

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a frame error packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame type is sent to the serial port for any type of frame error.

#### See also:

```
FrameError
XBeeAPIPacket
```

Class constructor. Instantiates a new FrameErrorPacket object with the provided parameters.

#### **Parameters**

- **frame\_error** (*FrameError*) the frame error.
- op\_mode (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# See also:

```
FrameError
XBeeAPIPacket
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns FrameErrorPacket

#### **Raises**

- InvalidPacketException if the bytearray length is less than 6. (start delim. + length (2 bytes) + frame type + frame error + checksum = 6 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).

- InvalidPacketException if the frame type is different from ApiFrameType. FRAME\_ERROR.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs_id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
error
    Returns the frame error of the packet.
```

**Returns** the frame error of the packet.

Return type FrameError

See also:

FrameError

# frame id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

factory

# get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
```

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a send data request packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame type is used to send a file of the given name and type to Device Cloud.

If the frame ID is non-zero, a SendDataResponsePacket will be received.

See also:

SendDataResponsePacket XBeeAPIPacket

Class constructor. Instantiates a new SendDataRequestPacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- path (String) path of the file to upload to Device Cloud.
- **content\_type** (*String*) content type of the file to upload.
- options (SendDataRequestOptions) the action when uploading a file.
- file\_data (Bytearray, optional) data of the file to upload.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *frame id* is less than 0 or greater than 255.

See also:

*XBeeAPIPacket* 

static create\_packet (raw, operating\_mode)

Override method.

**Returns** SendDataRequestPacket

### Raises

- InvalidPacketException if the bytearray length is less than 10. (start delim. + length (2 bytes) + frame type + frame id + path length + content type length + transport + options + checksum = 10 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)

- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. SEND DATA REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
     XBeeAPIPacket._check_api_packet()
needs id()
     Override method.
     See also:
     XBeeAPIPacket.needs id()
path
     Returns the path of the file to upload to Device Cloud.
         Returns the path of the file to upload to Device Cloud.
         Return type String
content_type
     Returns the content type of the file to upload.
         Returns the content type of the file to upload.
         Return type String
options
    Returns the file upload operation options.
         Returns the file upload operation options.
         Return type SendDataRequestOptions
```

SendDataRequestOptions

# file\_data

See also:

Returns the data of the file to upload.

**Returns** the data of the file to upload.

Return type Bytearray

```
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
```

# ApiFrameType

is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

### Return type Boolean

#### op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

### to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a send data response packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame type is sent out the serial port in response to the <code>SendDataRequestPacket</code>, providing its frame ID is non-zero.

See also:

SendDataRequestPacket XBeeAPIPacket

Class constructor. Instantiates a new SendDataResponsePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- status (DeviceCloudStatus) the file upload status.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

See also:

```
DeviceCloudStatus
XBeeAPIPacket
frame_id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
```

*ApiFrameType* 

#### is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

#### op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

### output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

**Return type** Bytearray

### to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

### static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

# static create\_packet (raw, operating\_mode)

Override method.

**Returns** SendDataResponsePacket

### Raises

- InvalidPacketException if the bytearray length is less than 10. (start delim. + length (2 bytes) + frame type + frame id + status + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. SEND\_DATA\_RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

See also:

XBeePacket.create packet()

```
xBeeAPIPacket._check_api_packet()
needs_id()
Override method.
See also:

xBeeAPIPacket.needs_id()

status
Returns the file upload status.
Return type DeviceCloudStatus
See also:

DeviceCloudStatus
```

### digi.xbee.packets.digimesh module

This class represents a DigiMesh Route Information packet. Packet is built using the parameters of the constructor or providing a valid API payload.

A Route Information Packet can be output for DigiMesh unicast transmissions on which the NACK enable or the Trace Route enable TX option is enabled.

See also:

*XBeeAPIPacket* 

Class constructor. Instantiates a new RouteInformationPacket object with the provided parameters.

#### **Parameters**

- src\_event (Integer) Source event identifier. 0x11=NACK, 0x12=Trace route
- **timestamp** (*Integer*) System timer value on the node generating the this packet. The timestamp is in microseconds.
- ack\_timeout\_count (Integer) The number of MAC ACK timeouts.

- **tx\_block\_count** (*Integer*) The number of times the transmission was blocked due to reception in progress.
- **dst\_addr** (XBee64BitAddress) The 64-bit address of the final destination node of this network-level transmission.
- **src\_addr** (XBee64BitAddress) The 64-bit address of the source node of this network-level transmission.
- **responder\_addr** (XBee64BitAddress) The 64-bit address of the node that generates this packet after it sends (or attempts to send) the packet to the next hop (successor node).
- **successor\_addr** (*XBee64BitAddress*) The 64-bit address of the next node after the responder in the route towards the destination, whether or not the packet arrived successfully at the successor node.
- additional\_data (Bytearray, optional, default=`None`) Additional data of the packet.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### Raises

- ValueError if src event is not 0x11 or 0x12.
- ValueError if timestamp is not between 0 and 0xFFFFFFFF.
- ValueError if *ack\_timeout\_count* or *tx\_block\_count* are not between 0 and 255.

### See also:

XBee64BitAddress XBeeAPIPacket

## static create\_packet (raw, operating\_mode)

Override method.

Returns RouteInformationPacket.

## Raises

- InvalidPacketException If the bytearray length is less than 46. (start delim. + length (2 bytes) + frame type + src\_event + length + timestamp (4 bytes) + ack timeout count + tx blocked count + reserved + dest addr (8 bytes) + src addr (8 bytes) + responder addr (8 bytes) + successor addr (8 bytes) + checksum = 46 bytes).
- $\bullet$  InvalidPacketException If the length field of  $\it raw$  is different from its real length. (length field: bytes 1 and 3)
- InvalidPacketException If the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException If the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException If the frame type is not ApiFrameType. DIGIMESH\_ROUTE\_INFORMATION.

- InvalidPacketException If the internal length byte of the rest of the frame (without the checksum) is different from its real length.
- InvalidOperatingModeException If operating\_mode is not supported.

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

## needs\_id()

Override method.

See also:

```
XBeeAPIPacket.needs_id()
```

### src\_event

Returns the source event.

**Returns** The source event.

Return type Integer

## length

Returns the number of bytes that follow, excluding the checksum.

Returns Data length.

Return type Integer

## timestamp

Returns the system timer value on the node generating this package. The timestamp is in microseconds.

**Returns** The system timer value in microseconds.

Return type Integer

## ack\_timeout\_count

Returns the number of MAC ACK timeouts that occur.

**Returns** The number of MAC ACK timeouts that occur.

Return type Integer

## tx\_block\_count

Returns the number of times the transmission was blocked due to reception in progress.

### **Returns**

The number of times the transmission was blocked due to reception in progress.

Return type Integer

# $frame\_id$

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
```

2.6. API reference 321

**Returns** The operating mode.

## Return type OperatingMode

### output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

### dst addr

Returns the 64-bit source address.

Returns

The 64-bit address of the final destination node.

Return type XBee64BitAddress

See also:

XBee64BitAddress

### src\_addr

Returns the 64-bit address of the source node of this network-level transmission.

**Returns** The 64-bit address of the source node.

Return type XBee64BitAddress

See also:

XBee64BitAddress

# ${\tt responder\_addr}$

Returns the 64-bit address of the node that generates this packet after it sends (or attempts to send) the packet to the next hop (successor node).

**Returns** The 64-bit address of the responder node.

Return type XBee64BitAddress

See also:

XBee64BitAddress

## successor\_addr

Returns the 64-bit address of the next node after the responder in the route towards the destination, whether or not the packet arrived successfully at the successor node.

**Returns** The 64-bit address of the successor node.

Return type XBee64BitAddress

See also:

XBee64BitAddress

## digi.xbee.packets.filesystem module

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a File System Request. Packet is built using the parameters of the constructor or providing a valid API payload.

A File System Request allows to access the filesystem and perform different operations.

Command response is received as an FSResponsePacket.

See also:

*XBeeAPIPacket* 

Class constructor. Instantiates a new FSRequestPacket object with the provided parameters.

### **Parameters**

- **frame\_id** (*Integer*) Frame ID of the packet.
- **command** (*FSCmd* or bytearray) File system command to execute.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError If *frame\_id* is less than 0 or greater than 255.
- TypeError If *command* is not a *FSCmd* or a bytearray.

See also:

FSCmd XBeeAPIPacket

### static create\_packet (raw, operating\_mode)

Override method.

Returns FSRequestPacket

### Raises

- InvalidPacketException If the bytearray length is less than 7 + the minimum length of the command. (start delim. + length (2 bytes) + frame type + frame id + fs cmd id + checksum + cmd data = 7 bytes + cmd data).
- InvalidPacketException If the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException If the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException If the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException If the frame type is different from ApiFrameType. FILE SYSTEM REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

#### needs id()

Override method.

## See also:

```
XBeeAPIPacket.needs_id()
```

### command

Returns the file system command of the packet.

**Returns** File system command of the packet.

Return type String

## frame id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

**Return type** Integer

## get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

**Returns** checksum value of this XBeePacket.

Return type Integer

```
See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
     ApiFrameType
is_broadcast()
    Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
    Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
```

## **Return type** Bytearray

```
to dict()
```

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

### static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a File System Response. Packet is built using the parameters of the constructor or providing a valid API payload.

This packet is received in response of an FSRequestPacket.

### See also:

XBeeAPIPacket

Class constructor. Instantiates a new FSResponsePacket object with the provided parameters.

## **Parameters**

- **frame\_id** (*Integer*) The frame ID of the packet.
- **command** (*FSCmd* or bytearray) File system command to execute.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError If *frame\_id* is less than 0 or greater than 255.
- TypeError If *command* is not a *FSCmd* or a bytearray.

## See also:

```
FSCmd
XBeeAPIPacket
```

```
static create_packet (raw, operating_mode)
```

Override method.

Returns FSResponsePacket

**Raises** 

- InvalidPacketException If the bytearray length is less than 8 + the minimum length of the command. (start delim. + length (2 bytes) + frame type + frame id + fs cmd id + status + checksum + cmd data = 8 bytes + cmd data).
- InvalidPacketException If the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException If the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException If the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException If the frame type is different from ApiFrameType. FILE\_SYSTEM\_RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()
```

### command

Returns the file system command of the packet.

**Returns** File system command of the packet.

Return type String

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

# ${\tt get\_checksum}\,(\,)$

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

factory

```
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get frame spec data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
```

Returns dictionary with all info of the XBeePacket fields.

328

## **Return type** Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a remote File System Request. Packet is built using the parameters of the constructor or providing a valid API payload.

Used to access the filesystem on a remote device and perform different operations.

Remote command options are set as a bitfield.

If configured, command response is received as a RemoteFSResponsePacket.

### See also:

```
RemoteFSResponsePacket
XBeeAPTPacket
```

Class constructor. Instantiates a new RemoteFSRequestPacket object with the provided parameters.

### **Parameters**

- **frame\_id** (*Integer*) Frame ID of the packet.
- **x64bit\_addr** (XBee64BitAddress) 64-bit destination address.
- **command** (FSCmd or bytearray) File system command to execute.
- transmit\_options (Integer, optional, default=`TransmitOptions. NONE.value`) Bitfield of supported transmission options.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# Raises

- ValueError If *frame\_id* is less than 0 or greater than 255.
- TypeError If command is not a FSCmd or a bytearray.

## See also:

FSCmd TransmitOptions XBee64BitAddress XBeeAPIPacket

```
static create_packet (raw, operating_mode)
```

Override method.

Returns RemoteFSRequestPacket

#### Raises

- InvalidPacketException If the bytearray length is less than 7 + the minimum length of the command. (start delim. + length (2 bytes) + frame type + frame id + 64bit addr. + transmit options + fs cmd id + checksum + cmd data = 16 bytes + cmd data).
- InvalidPacketException If the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException If the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException If the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException If the frame type is different from ApiFrameType. REMOTE FILE SYSTEM REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()
```

# ${\tt x64bit\_dest\_addr}$

Returns the 64-bit destination address.

**Returns** 64-bit destination address.

Return type XBee64BitAddress

See also:

XBee64BitAddress

## command

Returns the file system command of the packet.

**Returns** File system command of the packet.

**Return type** String

```
transmit_options
    Returns the transmit options bitfield.
         Returns Transmit options bitfield.
         Return type Integer
    See also:
     TransmitOptions
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
    See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
```

2.6. API reference 331

**Returns** the frame type integer value of this packet.

```
Return type Integer
```

ApiFrameType

### is broadcast()

Returns whether this packet is broadcast or not.

Returns True if this packet is broadcast, False otherwise.

Return type Boolean

### op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

## output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

**Return type** Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Remote File System Response. Packet is built using the parameters of the constructor or providing a valid API payload.

This packet is received in response of an RemoteFSRequestPacket.

See also:

RemoteFSRequestPacket XBeeAPIPacket Class constructor. Instantiates a new RemoteFSResponsePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) The frame ID of the packet.
- x64bit\_addr (XBee64BitAddress) 64-bit source address.
- **command** (FSCmd or bytearray) File system command to execute.
- **rx\_options** (*Integer*) Bitfield indicating the receive options.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError If *frame\_id* is less than 0 or greater than 255.
- TypeError If *command* is not a *FSCmd* or a bytearray.

### See also:

```
FSCmd
ReceiveOptions
XBeeAPIPacket
```

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

**Returns** checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

## get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

## get\_frame\_type()

Returns the frame type of this packet.

**Returns** the frame type of this packet.

```
Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
static create_packet (raw, operating_mode)
     Override method.
```

Returns RemoteFSResponsePacket

### Raises

- InvalidPacketException If the bytearray length is less than 8 + the minimum length of the command. (start delim. + length (2 bytes) + frame type + frame id + 64bit addr. + receive options + fs cmd id + status + checksum + cmd data = 17 bytes + cmd data).
- InvalidPacketException If the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException If the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException If the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException If the frame type is different from ApiFrameType. REMOTE\_FILE\_SYSTEM\_RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

x64bit_source_addr
Returns the 64-bit source address.
Returns 64-bit source address.
Return type XBee64BitAddress
See also:
```

## command

Returns the file system command of the packet.

Returns File system command of the packet.

Return type String

XBee64BitAddress

# receive\_options

Returns the receive options bitfield.

Returns Receive options bitfield.

## Return type Integer

#### See also:

ReceiveOptions

digi.xbee.packets.filesystem.build\_fs\_command(cmd\_bytearray, direction=0)
 Creates a file system command from raw data.

### **Parameters**

- cmd\_bytearray (Bytearray) Raw data of the packet to build.
- **direction** (Integer, optional, default=0) If this command is a request (0) or a response (1).

**Raises** InvalidPacketException – If *cmd\_bytearray* is not a bytearray or its length is less than 1 for requests 2 for responses.

See also:

**FSCmd** 

## digi.xbee.packets.network module

This class represents an RX (Receive) IPv4 packet. Packet is built using the parameters of the constructor or providing a valid byte array.

### See also:

```
TXIPv4Packet
XBeeAPIPacket
```

Class constructor. Instantiates a new RXIPv4Packet object with the provided parameters.

## **Parameters**

- src\_address (IPv4Address) IPv4 address of the source device.
- **dest\_port** (*Integer*) **destination** port number.
- **src\_port** (*Integer*) source port number.
- ip\_protocol (IPProtocol) IP protocol used for transmitted data.
- data (Bytearray, optional) data that is sent to the destination device.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if dest\_port is less than 0 or greater than 65535 or
- ValueError if *source\_port* is less than 0 or greater than 65535.

### See also:

*IPProtocol* 

## static create\_packet (raw, operating\_mode)

Override method.

**Returns** class: .RXIPv4Packet.

### Raises

- InvalidPacketException if the bytearray length is less than 15. (start delim + length (2 bytes) + frame type + source address(4 bytes) + dest port (2 bytes) + source port (2 bytes) + network protocol + status + checksum = 15 bytes)
- InvalidPacketException if the length field of *raw* is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of *raw* is not the header byte. See SPECIAL\_BYTE.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType.RX\_IPV4.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()
```

### source\_address

Returns the IPv4 address of the source device.

**Returns** the IPv4 address of the source device.

Return type ipaddress. IPv4Address

### dest\_port

Returns the destination port.

```
Returns the destination port.
         Return type Integer
source_port
    Returns the source port.
         Returns the source port.
         Return type Integer
ip_protocol
    Returns the IP protocol used for transmitted data.
         Returns the IP protocol used for transmitted data.
         Return type IPProtocol
data
     Returns the data of the packet.
         Returns the data of the packet.
         Return type Bytearray
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
```

```
ApiFrameType
```

```
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
```

```
Return type Bytearray

class digi.xbee.packets.network.TXIPv4Packet(frame_id, dest_address,
```

This class represents an TX (Transmit) IPv4 packet. Packet is built using the parameters of the constructor or providing a valid byte array.

See also:

RXIPv4Packet XBeeAPIPacket

Class constructor. Instantiates a new TXIPv4Packet object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID. Must be between 0 and 255.
- dest\_address (IPv4Address) IPv4 address of the destination device.
- **dest port** (*Integer*) destination port number.
- **src\_port** (*Integer*) source port number.
- ip\_protocol (IPProtocol) IP protocol used for transmitted data.
- tx opts (Integer) the transmit options of the packet.
- data (Bytearray, optional) data that is sent to the destination device.
- op\_mode (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *dest\_port* is less than 0 or greater than 65535.
- ValueError if *source\_port* is less than 0 or greater than 65535.

### See also:

*IPProtocol* 

# OPTIONS\_CLOSE\_SOCKET = 2

This option will close the socket after the transmission.

## OPTIONS\_LEAVE\_SOCKET\_OPEN = 0

This option will leave socket open after the transmission.

## static create\_packet (raw, operating\_mode)

Override method.

Returns TXIPv4Packet.

## Raises

- InvalidPacketException if the bytearray length is less than 16. (start delim + length (2 bytes) + frame type + frame id + dest address (4 bytes) + dest port (2 bytes) + source port (2 bytes) + network protocol + transmit options + checksum = 16 bytes)
- InvalidPacketException if the length field of *raw* is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of *raw* is not the header byte. See SPECIAL\_BYTE.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).

- InvalidPacketException if the frame type is not ApiFrameType.TX\_IPV4.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create packet()
    XBeeAPIPacket._check_api_packet()
needs id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
dest_address
    Returns the IPv4 address of the destination device.
        Returns the IPv4 address of the destination device.
        Return type ipaddress. IPv4Address
frame id
    Returns the frame ID of the packet.
        Returns the frame ID of the packet.
        Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
        Returns checksum value of this XBeePacket.
        Return type Integer
    See also:
    factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
```

2.6. API reference 341

Returns the frame type of this packet.

**Returns** the frame type of this packet.

```
Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
dest_port
     Returns the destination port.
```

**Returns** the destination port.

## Return type Integer

### source\_port

Returns the source port.

**Returns** the source port.

**Return type** Integer

## ip\_protocol

Returns the IP protocol used for transmitted data.

**Returns** the IP protocol used for transmitted data.

Return type IPProtocol

## transmit\_options

Returns the transmit options of the packet.

Returns the transmit options of the packet.

Return type Integer

### data

Returns the data of the packet.

**Returns** the data of the packet.

Return type Bytearray

## digi.xbee.packets.raw module

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a TX (Transmit) 64 Request packet. Packet is built using the parameters of the constructor or providing a valid byte array.

A TX Request message will cause the module to transmit data as an RF Packet.

## See also:

### XBeeAPIPacket

Class constructor. Instantiates a new TX64Packet object with the provided parameters.

## **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- x64bit\_addr (XBee64BitAddress) the 64-bit destination address.
- tx\_opts (Integer) bitfield of supported transmission options.
- rf\_data (Bytearray, optional) RF data that is sent to the destination device.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

See also:

```
TransmitOptions
XBee64BitAddress
XBeeAPIPacket
```

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

```
static create_packet (raw, operating_mode)
```

Override method.

Returns TX64Packet.

### **Raises**

- InvalidPacketException if the bytearray length is less than 15. (start delim. + length (2 bytes) + frame type + frame id + 64bit addr. + transmit options + checksum = 15 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. TX\_64.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

x64bit_dest_addr
Returns the 64-bit destination address.

Returns the 64-bit destination address.

Return type XBee64BitAddress
```

See also:

XBee64BitAddress

```
transmit_options
    Returns the transmit options bitfield.
         Returns the transmit options bitfield.
         Return type Integer
    See also:
     TransmitOptions
rf data
     Returns the RF data to send.
         Returns the RF data to send.
         Return type Bytearray
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
    ApiFrameType
```

```
get_frame_type_value()
```

Returns the frame type integer value of this packet.

**Returns** the frame type integer value of this packet.

Return type Integer

See also:

ApiFrameType

## is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

## op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

## output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a TX (Transmit) 16 Request packet. Packet is built using the parameters of the constructor or providing a valid byte array.

A TX request message will cause the module to transmit data as an RF packet.

See also:

*XBeeAPIPacket* 

Class constructor. Instantiates a new TX16Packet object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- x16bit\_addr (XBee16BitAddress) the 16-bit destination address.
- tx opts (Integer) bitfield of supported transmission options.
- rf\_data (Bytearray, optional) RF data that is sent to the destination device.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### See also:

```
TransmitOptions
XBee16BitAddress
XBeeAPIPacket
```

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

```
static create_packet (raw, operating_mode)
```

Override method.

Returns TX16Packet.

## Raises

- InvalidPacketException if the bytearray length is less than 9. (start delim. + length (2 bytes) + frame type + frame id + 16bit addr. + transmit options + checksum = 9 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. TX 16.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:
```

```
XBeeAPIPacket.needs_id()
x16bit_dest_addr
     Returns the 16-bit destination address.
         Returns the 16-bit destination address.
         Return type XBee16BitAddress
     See also:
     XBee16BitAddress
transmit_options
     Returns the transmit options bitfield.
         Returns the transmit options bitfield.
         Return type Integer
     See also:
     TransmitOptions
rf_data
     Returns the RF data to send.
         Returns the RF data to send.
         Return type Bytearray
frame_id
     Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
     Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
```

```
XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
```

Returns data unescaped.

Return type Bytearray

This class represents a TX (Transmit) status packet. Packet is built using the parameters of the constructor or providing a valid API payload.

When a TX request is completed, the module sends a TX status message. This message will indicate if the packet was transmitted successfully or if there was a failure.

### See also:

```
TX16Packet
TX64Packet
XBeeAPIPacket
```

Class constructor. Instantiates a new TXStatusPacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- tx\_status (TransmitStatus) transmit status.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

### See also:

```
TransmitStatus
XBeeAPIPacket
```

### static create\_packet (raw, operating\_mode)

Override method.

Returns TXStatusPacket.

## Raises

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + frame id + transmit status + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).

- InvalidPacketException if the frame type is different from ApiFrameType. TX\_16.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

transmit_status
Returns the transmit status.
Return type TransmitStatus
See also:

TransmitStatus
```

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

factory

# get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
```

## Returns data unescaped.

## Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an RX (Receive) 64 request packet. Packet is built using the parameters of the constructor or providing a valid API byte array.

When the module receives an RF packet, it is sent out the UART using this message type.

This packet is the response to TX (transmit) 64 request packets.

### See also:

ReceiveOptions TX64Packet XBeeAPIPacket

Class constructor. Instantiates a RX64Packet object with the provided parameters.

#### **Parameters**

- **x64bit\_addr** (*XBee64BitAddress*) the 64-bit source address.
- rssi (Integer) received signal strength indicator.
- rx\_opts (Integer) bitfield indicating the receive options.
- rf\_data(Bytearray, optional) received RF data.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

```
ReceiveOptions
XBee64BitAddress
XBeeAPIPacket
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns RX64Packet

## Raises

- InvalidPacketException if the bytearray length is less than 15. (start delim. + length (2 bytes) + frame type + 64bit addr. + rssi + receive options + checksum = 15 bytes).
- $\bullet$  InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)

- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. RX\_64.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
is_broadcast()
    Override method.
    See also:
    XBeeAPIPacket.is_broadcast()
x64bit source addr
    Returns the 64-bit source address.
        Returns the 64-bit source address.
        Return type XBee64BitAddress
    See also:
    XBee64BitAddress
rssi
    Returns the received Signal Strength Indicator (RSSI).
        Returns the received Signal Strength Indicator (RSSI).
        Return type Integer
receive_options
    Returns the receive options bitfield.
```

**Returns** the receive options bitfield.

```
Return type Integer
```

```
ReceiveOptions
```

## rf data

Returns the received RF data.

Returns the received RF data.

Return type Bytearray

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

# get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

# ${\tt get\_frame\_spec\_data}\:(\:)$

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

# get\_frame\_type()

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

```
ApiFrameType
```

# get\_frame\_type\_value()

Returns the frame type integer value of this packet.

```
Returns the frame type integer value of this packet.
```

Return type Integer

See also:

ApiFrameType

# op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

```
static unescape_data(data)
```

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an RX (Receive) 16 Request packet. Packet is built using the parameters of the constructor or providing a valid API byte array.

When the module receives an RF packet, it is sent out the UART using this message type

This packet is the response to TX (Transmit) 16 Request packets.

See also:

ReceiveOptions TX16Packet XBeeAPIPacket

Class constructor. Instantiates a RX16Packet object with the provided parameters.

## **Parameters**

- x16bit\_addr (XBee16BitAddress) the 16-bit source address.
- rssi (Integer) received signal strength indicator.
- rx\_opts (Integer) bitfield indicating the receive options.
- rf\_data (Bytearray, optional) received RF data.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

```
ReceiveOptions
XBee16BitAddress
XBeeAPIPacket
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns RX16Packet.

#### Raises

- InvalidPacketException if the bytearray length is less than 9.
- (start delim. + length (2 bytes) + frame type + 16bit addr. + rssi + receive options + checksum = 9 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. RX\_16.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs id()
```

```
is broadcast()
    Override method.
     See also:
     XBeeAPIPacket.is broadcast()
x16bit_source_addr
    Returns the 16-bit source address.
         Returns the 16-bit source address.
         Return type XBee16BitAddress
    See also:
     XBee16BitAddress
rssi
     Returns the received Signal Strength Indicator (RSSI).
         Returns the received Signal Strength Indicator (RSSI).
         Return type Integer
receive_options
     Returns the receive options bitfield.
         Returns the receive options bitfield.
         Return type Integer
     See also:
     ReceiveOptions
rf data
    Returns the received RF data.
         Returns the received RF data.
         Return type Bytearray
frame_id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes

between the length field and the checksum field.

**Return type** Integer

Returns checksum value of this XBeePacket.

```
See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
     ApiFrameType
op_mode
    Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
    Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
    Returns a dictionary with all information of the XBeePacket fields.
```

2.6. API reference 359

**Returns** dictionary with all info of the XBeePacket fields.

# **Return type** Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents an RX64 address IO packet. Packet is built using the parameters of the constructor or providing a valid API payload.

I/O data is sent out the UART using an API frame.

## See also:

XBeeAPIPacket

Class constructor. Instantiates an RX64IOPacket object with the provided parameters.

## **Parameters**

- x64bit addr (XBee64BitAddress) the 64-bit source address.
- rssi (Integer) received signal strength indicator.
- rx\_opts (Integer) bitfield indicating the receive options.
- data (Bytearray) received RF data.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# See also:

```
ReceiveOptions
XBee64BitAddress
XBeeAPIPacket
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns RX64IOPacket.

## Raises

- InvalidPacketException if the bytearray length is less than 20. (start delim. + length (2 bytes) + frame type + 64bit addr. + rssi + receive options + rf data (5 bytes) + checksum = 20 bytes)
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)

- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. RX\_IO\_64.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
is_broadcast()
    Override method.
    See also:
    XBeeAPIPacket.is_broadcast()
x64bit source addr
    Returns the 64-bit source address.
        Returns the 64-bit source address.
        Return type XBee64BitAddress
    See also:
    XBee64BitAddress
rssi
    Returns the received Signal Strength Indicator (RSSI).
        Returns the received Signal Strength Indicator (RSSI).
        Return type Integer
receive_options
    Returns the receive options bitfield.
```

**Returns** the receive options bitfield.

```
Return type Integer See also:
```

```
ReceiveOptions
```

## rf data

Returns the received RF data.

Returns the received RF data.

Return type Bytearray

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

# get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

# ${\tt get\_frame\_spec\_data}\:(\:)$

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

# get\_frame\_type()

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

```
ApiFrameType
```

# get\_frame\_type\_value()

Returns the frame type integer value of this packet.

```
Return type Integer
          See also:
          ApiFrameType
     op_mode
          Retrieves the operating mode in which this packet was read.
              Returns The operating mode.
              Return type OperatingMode
     output (escaped=False)
          Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
              Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
              Returns raw bytearray of the XBeePacket.
              Return type Bytearray
     to_dict()
          Returns a dictionary with all information of the XBeePacket fields.
              Returns dictionary with all info of the XBeePacket fields.
              Return type Dictionary
     static unescape_data(data)
          Un-escapes the provided bytearray data.
              Parameters data (Bytearray) – the bytearray to unescape.
              Returns data unescaped.
              Return type Bytearray
     io sample
          Returns the IO sample corresponding to the data contained in the packet.
              Returns
                  the IO sample of the packet, None if the packet has not any data or if the sample could not
                    be generated correctly.
              Return type IOSample
          See also:
          IOSample
class digi.xbee.packets.raw.RX16IOPacket (x16bit_addr,
                                                                         rssi,
                                                                                                data,
                                                                                   rx_opts,
                                                       op_mode=<OperatingMode.API_MODE:
                                                                                                  (1,
                                                        'API mode')>)
     Bases: digi.xbee.packets.base.XBeeAPIPacket
     This class represents an RX16 address IO packet. Packet is built using the parameters of the constructor or
     providing a valid byte array.
```

**Returns** the frame type integer value of this packet.

I/O data is sent out the UART using an API frame.

## See also:

```
XBeeAPIPacket
```

Class constructor. Instantiates an RX16IOPacket object with the provided parameters.

## **Parameters**

- x16bit\_addr (XBee16BitAddress) the 16-bit source address.
- rssi (Integer) received signal strength indicator.
- **rx\_opts** (*Integer*) bitfield indicating the receive options.
- data (Bytearray) received RF data.
- op\_mode (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

```
ReceiveOptions
XBee16BitAddress
XBeeAPIPacket
```

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

```
get_frame_spec_data()
```

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
static create_packet (raw, operating_mode)
     Override method.
         Returns RX16IOPacket.
         Raises
```

- InvalidPacketException if the bytearray length is less than 14. (start delim. + length (2 bytes) + frame type + 16bit addr. + rssi + receive options + rf data (5 bytes) + checksum = 14 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is different from ApiFrameType. RX\_IO\_16.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs_id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
is_broadcast()
    Override method.
    See also:
    XBeeAPIPacket.is_broadcast()
x16bit_source_addr
    Returns the 16-bit source address.
        Returns the 16-bit source address.
        Return type XBee16BitAddress
    See also:
    XBee16BitAddress
```

#### rssi

Returns the received Signal Strength Indicator (RSSI).

Returns the received Signal Strength Indicator (RSSI).

## Return type Integer

# receive\_options

Returns the receive options bitfield.

**Returns** the receive options bitfield.

**Return type** Integer

See also:

ReceiveOptions

## rf\_data

Returns the received RF data.

Returns the received RF data.

Return type Bytearray

## io\_sample

Returns the IO sample corresponding to the data contained in the packet.

#### Returns

**the IO sample of the packet,** *None* **if the** packet has not any data or if the sample could not be generated correctly.

Return type IOSample

See also:

*IOSample* 

# digi.xbee.packets.relay module

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a User Data Relay packet. Packet is built using the parameters of the constructor.

The User Data Relay packet allows for data to come in on an interface with a designation of the target interface for the data to be output on.

The destination interface must be one of the interfaces found in the corresponding enumerator (see XBeeLocalInterface).

See also:

UserDataRelayOutputPacket
XBeeAPIPacket
XBeeLocalInterface

Class constructor. Instantiates a new UserDataRelayPacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*integer*) the frame ID of the packet.
- local\_iface (XBeeLocalInterface) the destination interface.
- data (Bytearray, optional) Data to send to the destination interface.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

```
XBeeAPIPacket
XBeeLocalInterface
```

#### Raises

- ValueError if *local\_interface* is *None*.
- ValueError if *frame\_id* is less than 0 or greater than 255.

# static create\_packet (raw, operating\_mode)

Override method.

Returns UserDataRelayPacket.

### Raises

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + frame id + relay interface + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- $\bullet$  InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. USER\_DATA\_RELAY\_REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

# See also:

```
XBeeAPIPacket.needs id()
data
    Returns the data to send.
        Returns the data to send.
        Return type Bytearray
dest_interface
    Returns the the destination interface.
        Returns the destination interface.
        Return type XBeeLocalInterface
    See also:
    XBeeLocalInterface
frame_id
    Returns the frame ID of the packet.
        Returns the frame ID of the packet.
        Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
        Returns checksum value of this XBeePacket.
        Return type Integer
    See also:
    factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
```

2.6. API reference 369

get\_frame\_type()

See also:

Returns the frame type of this packet.

Return type ApiFrameType

**Returns** the frame type of this packet.

```
ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape data(data)
     Un-escapes the provided bytearray data.
```

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a User Data Relay Output packet. Packet is built using the parameters of the constructor.

The User Data Relay Output packet can be received from any relay interface.

The source interface must be one of the interfaces found in the corresponding enumerator (see XBeeLocalInterface).

UserDataRelayPacket XBeeAPIPacket XBeeLocalInterface

Class constructor. Instantiates a new UserDataRelayOutputPacket object with the provided parameters.

#### **Parameters**

- local\_iface (XBeeLocalInterface) the source interface.
- data (Bytearray, optional) Data received from the source interface.
- op\_mode (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *local\_interface* is *None*.

# See also:

XBeeAPIPacket XBeeLocalInterface

## static create\_packet (raw, operating\_mode)

Override method.

Returns UserDataRelayOutputPacket.

## Raises

- InvalidPacketException if the bytearray length is less than 6. (start delim. + length (2 bytes) + frame type + relay interface + checksum = 6 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. USER\_DATA\_RELAY\_OUTPUT.
- InvalidOperatingModeException if operating\_mode is not supported.

# See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

```
needs id()
    Override method.
     See also:
     XBeeAPIPacket.needs id()
frame_id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
```

# ApiFrameType

# is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

# op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

# data

Returns the received data.

**Returns** the received data.

Return type Bytearray

# src interface

Returns the the source interface.

**Returns** the source interface.

Return type XBeeLocalInterface

See also:

XBeeLocalInterface

# digi.xbee.packets.socket module

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Create packet. Packet is built using the parameters of the constructor.

Use this frame to create a new socket with the following protocols: TCP, UDP, or TLS.

#### See also:

SocketCreateResponsePacket XBeeAPIPacket

Class constructor. Instantiates a new SocketCreatePacket object with the provided parameters.

# **Parameters**

- frame id (Integer) the frame ID of the packet.
- **protocol** (*IPProtocol*) the protocol used to create the socket.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

XBeeAPIPacket IPProtocol

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

static create\_packet (raw, operating\_mode)

Override method.

Returns SocketCreatePacket.

# Raises

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + frame id + protocol + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET CREATE.
- InvalidOperatingModeException if operating\_mode is not supported.

```
See also:
    XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs_id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
protocol
    Returns the communication protocol.
        Returns the communication protocol.
        Return type IPProtocol
    See also:
     IPProtocol
frame id
    Returns the frame ID of the packet.
        Returns the frame ID of the packet.
        Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
        Returns checksum value of this XBeePacket.
        Return type Integer
     See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Create Response packet. Packet is built using the parameters of the constructor.

The device sends this frame in response to a Socket Create (0x40) frame. It contains a socket ID that should be used for future transactions with the socket and a status field.

If the status field is non-zero, which indicates an error, the socket ID will be set to 0xFF and the socket will not be opened.

# See also:

```
SocketCreatePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketCreateResponsePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the unique socket ID to address the socket.
- **status** (*SocketStatus*) the socket create status.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# See also:

```
XBeeAPIPacket
SocketStatus
```

# Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket id* is less than 0 or greater than 255.

# static create\_packet (raw, operating\_mode)

Override method.

Returns SocketCreateResponsePacket.

# Raises

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + socket id + status + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.

- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_CREATE\_RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

socket_id
Returns the socket ID.
Returns the socket ID.
```

#### status

Returns the socket create status.

Return type Integer

Returns the status.

Return type SocketStatus

See also:

SocketStatus

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

# get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         \textbf{Return type} \ \textit{OperatingMode}
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
```

```
to dict()
```

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

## static unescape data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Option Request packet. Packet is built using the parameters of the constructor.

Use this frame to modify the behavior of sockets to be different from the normal default behavior.

If the Option Data field is zero-length, the Socket Option Response Packet (0xC1) reports the current effective value.

## See also:

```
SocketOptionResponsePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketOptionRequestPacket object with the provided parameters.

## Parameters

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the socket ID to modify.
- **option** (*SocketOption*) the socket option of the parameter to change.
- option\_data (Bytearray, optional) the option data.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# See also:

```
XBeeAPIPacket
SocketOption
```

# Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.

```
static create_packet (raw, operating_mode)
```

Override method.

Returns SocketOptionRequestPacket.

## Raises

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + socket id + option + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: byte 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_OPTION\_REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create packet()
    XBeeAPIPacket._check_api_packet()
needs id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
socket id
    Returns the the socket ID.
        Returns the socket ID.
        Return type Integer
option
    Returns the socket option.
        Returns the socket option.
        Return type SocketOption
    See also:
    SocketOption
```

# option\_data

Returns the socket option data.

```
Returns the socket option data.
         Return type Bytearray
frame_id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
    See also:
     ApiFrameType
```

#### is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

# op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

on\_add=None,
op\_mode=<OperatingMode.API\_MODE:
(1, 'API mode')>)

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Option Response packet. Packet is built using the parameters of the constructor.

Reports the status of requests made with the Socket Option Request (0x41) packet.

See also:

```
SocketOptionRequestPacket
XBeeAPIPacket
```

Class constructor. Instantiates a new <code>SocketOptionResponsePacket</code> object with the provided parameters.

### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the socket ID for which modification was requested.

- **option** (SocketOption) the socket option of the parameter requested.
- **status** (*SocketStatus*) the socket option status of the parameter requested.
- option\_data (Bytearray, optional) the option data.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

```
XBeeAPIPacket
SocketOption
SocketStatus
```

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.

# static create\_packet (raw, operating\_mode)

Override method.

Returns SocketOptionResponsePacket.

## Raises

- InvalidPacketException if the bytearray length is less than 9. (start delim. + length (2 bytes) + frame type + frame id + socket id + option + status + checksum = 9 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_OPTION\_RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

# See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
See also:
```

XBeeAPIPacket.needs\_id()

## socket id

Returns the the socket ID.

**Returns** the socket ID.

Return type Integer

# option

Returns the socket option.

Returns the socket option.

Return type SocketOption

See also:

SocketOption

#### status

Returns the socket option status.

**Returns** the socket option status.

Return type SocketStatus

See also:

SocketStatus

# option\_data

Returns the socket option data.

Returns the socket option data.

Return type Bytearray

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

## get checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

factory

```
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get frame spec data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
```

**Return type** Dictionary

```
static unescape_data(data)
```

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Connect packet. Packet is built using the parameters of the constructor.

Use this frame to create a socket connect message that causes the device to connect a socket to the given address and port.

For a UDP socket, this filters out any received responses that are not from the specified remote address and port.

Two frames occur in response:

- Socket Connect Response frame (SocketConnectResponsePacket): Arrives immediately and confirms the request.
- Socket Status frame (SocketStatePacket): Indicates if the connection was successful.

#### See also:

```
SocketConnectResponsePacket
SocketStatePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketConnectPacket object with the provided parameters.

# **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the ID of the socket to connect.
- **dest port** (*Integer*) the destination port number.
- dest\_address\_type (Integer) the destination address type.
   One of SocketConnectPacket.DEST\_ADDRESS\_BINARY or SocketConnectPacket.DEST\_ADDRESS\_STRING.
- dest\_address (Bytearray or String) the destination address.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

```
SocketConnectPacket.DEST_ADDRESS_BINARY
SocketConnectPacket.DEST_ADDRESS_STRING
```

### *XBeeAPIPacket*

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.
- ValueError if *dest\_port* is less than 0 or greater than 65535.
- ValueError if dest\_address\_type is different than SocketConnectPacket.

  DEST\_ADDRESS\_BINARY and SocketConnectPacket.

  DEST\_ADDRESS\_STRING.
- ValueError if *dest\_address* is *None* or does not follow the format specified in the configured type.

# DEST\_ADDRESS\_BINARY = 0

Indicates the destination address field is a binary IPv4 address in network byte order.

### DEST ADDRESS STRING = 1

Indicates the destination address field is a string containing either a dotted quad value or a domain name to be resolved.

# static create\_packet (raw, operating\_mode)

Override method.

Returns SocketConnectPacket.

#### Raises

- InvalidPacketException if the bytearray length is less than 11. (start delim. + length (2 bytes) + frame type + frame id + socket id + dest port (2 bytes) + dest address type + dest\_address + checksum = 11 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET CONNECT.
- InvalidOperatingModeException if operating mode is not supported.

# See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

# needs\_id()

Override method.

# See also:

```
XBeeAPIPacket.needs id()
socket_id
    Returns the the socket ID.
         Returns the socket ID.
         Return type Integer
dest_port
    Returns the destination port.
         Returns the destination port.
         Return type Integer
dest_address_type
    Returns the destination address type.
         Returns the destination address type.
         Return type Integer
dest address
    Returns the destination address.
         Returns the destination address.
         Return type Bytearray or String
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
```

2.6. API reference 389

Returns the frame type of this packet.

```
Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Connect Response packet. Packet is built using the parameters of the constructor.

The device sends this frame in response to a Socket Connect (0x42) frame. The frame contains a status regarding the initiation of the connect.

#### See also:

SocketConnectPacket XBeeAPIPacket

Class constructor. Instantiates a new SocketConnectPacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the ID of the socket to connect.
- **status** (*SocketStatus*) the socket connect status.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

XBeeAPIPacket SocketStatus

## Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.

static create\_packet (raw, operating\_mode)

Override method.

Returns SocketConnectResponsePacket.

### Raises

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + socket id + status + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).

- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_CONNECT\_RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:
```

```
XBeeAPIPacket.needs_id()
```

## socket\_id

Returns the the socket ID.

**Returns** the socket ID.

Return type Integer

### status

Returns the socket connect status.

**Returns** the socket connect status.

Return type SocketStatus

See also:

```
SocketStatus
```

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

## get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

**Returns** checksum value of this XBeePacket.

Return type Integer

See also:

factory

```
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get frame spec data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
```

### **Return type** Dictionary

### static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Close packet. Packet is built using the parameters of the constructor.

Use this frame to close a socket when given an identifier.

#### See also:

```
SocketCloseResponsePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketClosePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- $socket\_id(Integer)$  the ID of the socket to close.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

*XBeeAPIPacket* 

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.

## static create\_packet (raw, operating\_mode)

Override method.

Returns SocketClosePacket.

### Raises

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + frame id + socket id + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)

- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_CLOSE.
- InvalidOperatingModeException if operating\_mode is not supported.

See also:

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
socket_id
    Returns the socket ID.
        Returns the socket ID.
        Return type Integer
frame_id
    Returns the frame ID of the packet.
        Returns the frame ID of the packet.
        Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
        Returns checksum value of this XBeePacket.
        Return type Integer
    See also:
    factory
get_frame_spec_data()
    Override method.
```

```
XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
```

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Close Response packet. Packet is built using the parameters of the constructor.

The device sends this frame in response to a Socket Close (0x43) frame. Since a close will always succeed for a socket that exists, the status can be only one of two values:

- · Success.
- · Bad socket ID.

#### See also:

```
SocketClosePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketCloseResponsePacket object with the provided parameters.

### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the ID of the socket to close.
- **status** (*SocketStatus*) the socket close status.
- op\_mode (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

```
XBeeAPIPacket
SocketStatus
```

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.

static create\_packet (raw, operating\_mode)

Override method.

 $\textbf{Returns} \ \textit{SocketCloseResponsePacket}.$ 

### Raises

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + socket id + status + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)

- InvalidPacketException if the first byte of 'raw' is not the header byte. See Special Byte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET CLOSE RESPONSE.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
    XBeeAPIPacket._check_api_packet()
needs id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
socket_id
    Returns the the socket ID.
```

**Returns** the socket ID.

**Return type** Integer

## status

Returns the socket close status.

**Returns** the socket close status.

Return type SocketStatus

See also:

SocketStatus

## frame\_id

398

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

**Return type** Integer

## get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

```
See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
    Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
    Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
```

## Return type Bytearray

```
to_dict()
```

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

```
static unescape_data(data)
```

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Send packet. Packet is built using the parameters of the constructor.

A Socket Send message causes the device to transmit data using the current connection. For a nonzero frame ID, this will elicit a Transmit (TX) Status - 0x89 frame (TransmitStatusPacket).

This frame requires a successful Socket Connect - 0x42 frame first (SocketConnectPacket). For a socket that is not connected, the device responds with a Transmit (TX) Status - 0x89 frame with an error.

#### See also:

TransmitStatusPacket
XBeeAPIPacket

Class constructor. Instantiates a new SocketSendPacket object with the provided parameters.

### **Parameters**

- $frame_id(Integer)$  the frame ID of the packet.
- **socket id** (*Integer*) the socket identifier.
- payload (Bytearray, optional) data that is sent.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.

## See also:

*XBeeAPIPacket* 

## static create\_packet (raw, operating\_mode)

Override method.

Returns SocketSendPacket.

#### Raises

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + frame id + socket ID + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_SEND.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

frame id

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

socket_id
Returns the socket ID.
Returns the socket ID.
Return type Integer

payload
Returns the payload to send.
Return type Bytearray
```

Returns the frame ID of the packet.

Return type Integer

**Returns** the frame ID of the packet.

```
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
```

**Returns** The operating mode.

### Return type OperatingMode

```
output (escaped=False)
```

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

Return type Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Send packet. Packet is built using the parameters of the constructor.

A Socket SendTo (Transmit Explicit Data) message causes the device to transmit data using an IPv4 address and port. For a non-zero frame ID, this will elicit a Transmit (TX) Status - 0x89 frame (TransmitStatusPacket).

If this frame is used with a TCP, SSL, or a connected UDP socket, the address and port fields are ignored.

## See also:

```
TransmitStatusPacket
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketSendToPacket object with the provided parameters.

## **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the socket identifier.
- **dest\_address** (IPv4Address) IPv4 address of the destination device.
- **dest\_port** (*Integer*) **destination** port number.
- payload (Bytearray, optional) data that is sent.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### **Raises**

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if socket\_id is less than 0 or greater than 255.
- ValueError if *dest\_port* is less than 0 or greater than 65535.

XBeeAPIPacket

## static create\_packet (raw, operating\_mode)

Override method.

Returns SocketSendToPacket.

### Raises

- InvalidPacketException if the bytearray length is less than 14. (start delim. + length (2 bytes) + frame type + frame id + socket ID + dest address (4 bytes) + dest port (2 bytes) + transmit options + checksum = 14 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_SENDTO.
- InvalidOperatingModeException if operating\_mode is not supported.

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

### needs\_id()

Override method.

## See also:

```
XBeeAPIPacket.needs_id()
```

## socket\_id

Returns the socket ID.

**Returns** the socket ID.

Return type Integer

### dest\_address

Returns the IPv4 address of the destination device.

```
Returns the IPv4 address of the destination device.
         Return type ipaddress. IPv4Address
dest_port
     Returns the destination port.
         Returns the destination port.
         Return type Integer
payload
     Returns the payload to send.
         Returns the payload to send.
         Return type Bytearray
frame_id
     Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
     Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
```

```
Returns the frame type integer value of this packet.
```

Return type Integer

See also:

ApiFrameType

## is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

### op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

## output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Bind/Listen packet. Packet is built using the parameters of the constructor.

Opens a listener socket that listens for incoming connections.

When there is an incoming connection on the listener socket, a Socket New IPv4 Client - 0xCC frame (SocketNewIPv4ClientPacket) is sent, indicating the socket ID for the new connection along with the remote address information.

For a UDP socket, this frame binds the socket to a given port. A bound UDP socket can receive data with a Socket Receive From: IPv4 - 0xCE frame (SocketReceiveFromIPv4Packet).

See also:

SocketNewIPv4ClientPacket
SocketReceiveFromIPv4Packet
XBeeAPIPacket

Class constructor. Instantiates a new SocketBindListenPacket object with the provided parameters.

### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- socket\_id (Integer) socket ID to listen on.
- **src\_port** (*Integer*) the port to listen on.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.
- ValueError if *source\_port* is less than 0 or greater than 65535.

#### See also:

XBeeAPIPacket

## static create\_packet (raw, operating\_mode)

Override method.

Returns SocketBindListenPacket.

### Raises

- InvalidPacketException if the bytearray length is less than 9. (start delim. + length (2 bytes) + frame type + frame id + socket ID + source port (2 bytes) + checksum = 9 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_BIND.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

```
needs id()
    Override method.
     See also:
     XBeeAPIPacket.needs id()
socket_id
    Returns the socket ID.
         Returns the socket ID.
         Return type Integer
source_port
    Returns the source port.
         Returns the source port.
         Return type Integer
frame_id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
     Override method.
     See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
```

```
ApiFrameType
```

device

sends

(SocketBindListenPacket).

this

frame

in

```
get_frame_type_value()
          Returns the frame type integer value of this packet.
              Returns the frame type integer value of this packet.
              Return type Integer
          See also:
          ApiFrameType
     is_broadcast()
          Returns whether this packet is broadcast or not.
              Returns True if this packet is broadcast, False otherwise.
              Return type Boolean
     op_mode
          Retrieves the operating mode in which this packet was read.
              Returns The operating mode.
              Return type OperatingMode
     output (escaped=False)
          Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
              Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
              Returns raw bytearray of the XBeePacket.
              Return type Bytearray
     to_dict()
          Returns a dictionary with all information of the XBeePacket fields.
              Returns dictionary with all info of the XBeePacket fields.
              Return type Dictionary
     static unescape_data(data)
          Un-escapes the provided bytearray data.
              Parameters data (Bytearray) – the bytearray to unescape.
              Returns data unescaped.
              Return type Bytearray
class digi.xbee.packets.socket.SocketListenResponsePacket (frame_id,
                                                                              socket id.
                                                                                              status,
                                                                              op_mode=<OperatingMode.API_MODE:
                                                                              (1, 'API mode')>)
     Bases: digi.xbee.packets.base.XBeeAPIPacket
     This class represents a Socket Listen Response packet. Packet is built using the parameters of the constructor.
```

2.6. API reference 409

response

Socket

Bind/Listen

(0x46)

frame

SocketBindListenPacket XBeeAPIPacket

Class constructor. Instantiates a new SocketListenResponsePacket object with the provided parameters.

### **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- socket\_id(Integer) socket ID.
- status (SocketStatus) socket listen status.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.

#### See also:

XBeeAPIPacket SocketStatus

## static create\_packet (raw, operating\_mode)

Override method.

Returns SocketListenResponsePacket.

### **Raises**

- InvalidPacketException if the bytearray length is less than 8. (start delim. + length (2 bytes) + frame type + frame id + socket ID + status + checksum = 8 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- $\bullet$  InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_LISTEN\_RESPONSE.
- InvalidOperatingModeException if operating mode is not supported.

## See also:

XBeePacket.create\_packet()

```
XBeeAPIPacket._check_api_packet()
needs_id()
    Override method.
    See also:
    XBeeAPIPacket.needs_id()
socket_id
    Returns the socket ID.
         Returns the socket ID.
         Return type Integer
status
    Returns the socket listen status.
         Returns The socket listen status.
         Return type SocketStatus
    See also:
     SocketStatus
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
```

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket New IPv4 Client packet. Packet is built using the parameters of the constructor.

XBee Cellular modem uses this frame when an incoming connection is accepted on a listener socket.

This frame contains the original listener's socket ID and a new socket ID of the incoming connection, along with the connection's remote address information.

#### See also:

XBeeAPIPacket.

Class constructor. Instantiates a new SocketNewIPv4ClientPacket object with the provided parameters.

#### **Parameters**

- **socket\_id** (*Integer*) the socket ID of the listener socket.
- client\_socket\_id (Integer) the socket ID of the new connection.
- remote\_address (IPv4Address) the remote IPv4 address.
- remote\_port (Integer) the remote port number.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *socket\_id* is less than 0 or greater than 255.
- ValueError if *client\_socket\_id* is less than 0 or greater than 255.
- ValueError if *remote\_port* is less than 0 or greater than 65535.

## See also:

*XBeeAPIPacket* 

```
static create_packet (raw, operating_mode)
```

Override method.

Returns SocketNewIPv4ClientPacket.

### Raises

- InvalidPacketException if the bytearray length is less than 13. (start delim. + length (2 bytes) + frame type + socket ID + client socket ID + remote address (4 bytes) + remote port (2 bytes) + checksum = 13 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)

- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET NEW IPV4 CLIENT.
- InvalidOperatingModeException if operating\_mode is not supported.

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
See also:
```

```
XBeeAPIPacket.needs_id()
```

## socket\_id

Returns the socket ID.

**Returns** the socket ID.

Return type Integer

## client\_socket\_id

Returns the client socket ID.

**Returns** the client socket ID.

Return type Integer

## remote\_address

Returns the remote IPv4 address.

**Returns** the remote IPv4 address.

Return type ipaddress. IPv4Address

## remote\_port

Returns the remote port.

**Returns** the remote port.

Return type Integer

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
     See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
    Retrieves the operating mode in which this packet was read.
```

2.6. API reference 415

**Returns** The operating mode.

## Return type OperatingMode

```
output (escaped=False)
```

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

Return type Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Receive packet. Packet is built using the parameters of the constructor.

XBee Cellular modem uses this frame when it receives RF data on the specified socket.

#### See also:

```
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketReceivePacket object with the provided parameters.

## **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- **socket\_id** (*Integer*) the ID of the socket the data has been received on.
- payload (Bytearray, optional) data that is received.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket id* is less than 0 or greater than 255.

## See also:

XBeeAPIPacket

```
frame id
     Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get checksum()
     Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
     See also:
     factory
```

Returns checksum value of this XBeePacket.

Return type Integer

```
get_frame_spec_data()
```

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
```

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

ApiFrameType

## get\_frame\_type\_value()

Returns the frame type integer value of this packet.

**Returns** the frame type integer value of this packet.

Return type Integer

See also:

ApiFrameType

## is\_broadcast()

Returns whether this packet is broadcast or not.

Returns True if this packet is broadcast, False otherwise.

### Return type Boolean

### op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

### output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

## static create\_packet (raw, operating\_mode)

Override method.

**Returns** SocketReceivePacket.

### Raises

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + frame id + socket ID + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_RECEIVE.
- $\bullet \ \, {\tt InvalidOperatingModeException-if} \ \textit{operating\_mode} \ \text{is not supported}.$

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

```
needs id()
```

Override method.

See also:

```
XBeeAPIPacket.needs id()
```

### socket id

Returns the socket ID.

**Returns** the socket ID.

**Return type** Integer

## payload

Returns the payload that was received.

**Returns** the payload that was received.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Socket Receive From packet. Packet is built using the parameters of the constructor.

XBee Cellular modem uses this frame when it receives RF data on the specified socket. The frame also contains addressing information about the source.

## See also:

```
XBeeAPIPacket
```

Class constructor. Instantiates a new SocketReceiveFromPacket object with the provided parameters.

## **Parameters**

- **frame\_id** (*Integer*) the frame ID of the packet.
- socket\_id (Integer) the ID of the socket the data has been received on.
- src\_address (IPv4Address) IPv4 address of the source device.
- **src\_port** (*Integer*) source port number.
- payload (Bytearray, optional) data that is received.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if *socket\_id* is less than 0 or greater than 255.
- ValueError if *source\_port* is less than 0 or greater than 65535.

```
See also:
XBeeAPIPacket
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
```

420

ApiFrameType

### is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

## op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

#### output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

## static create\_packet(raw, operating\_mode)

Override method.

Returns SocketReceiveFromPacket.

## Raises

- InvalidPacketException if the bytearray length is less than 13. (start delim. + length (2 bytes) + frame type + frame id + socket ID + source address (4 bytes) + source port (2 bytes) + status + Checksum = 14 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType.
   SOCKET\_RECEIVE\_FROM.
- InvalidOperatingModeException if operating\_mode is not supported.

See also:

```
XBeePacket.create_packet()
          XBeeAPIPacket._check_api_packet()
     needs_id()
          Override method.
          See also:
          XBeeAPIPacket.needs_id()
     socket id
          Returns the socket ID.
              Returns the socket ID.
              Return type Integer
     source address
          Returns the IPv4 address of the source device.
              Returns the IPv4 address of the source device.
              Return type ipaddress. IPv4Address
     source_port
          Returns the source port.
              Returns the source port.
              Return type Integer
     payload
          Returns the payload to send.
              Returns the payload that has been received.
              Return type Bytearray
class digi.xbee.packets.socket.SocketStatePacket (socket_id,
                                                                                             state,
                                                                 op_mode=<OperatingMode.API_MODE:</pre>
                                                                 (1, 'API mode')>)
     Bases: digi.xbee.packets.base.XBeeAPIPacket
     This class represents a Socket State packet. Packet is built using the parameters of the constructor.
     This frame is sent out the device's serial port to indicate the state related to the socket.
     See also:
     XBeeAPIPacket
     Class constructor. Instantiates a new SocketStatePacket object with the provided parameters.
          Parameters
                • socket_id (Integer) – the socket identifier.
```

• **state** (SocketState) – socket status.

• **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') – The mode in which the frame was captured.

**Raises** ValueError – if *socket\_id* is less than 0 or greater than 255.

See also:

```
SockeState 
XBeeAPIPacket
```

## frame\_id

Returns the frame ID of the packet.

Returns the frame ID of the packet.

Return type Integer

## get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

# get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

## get\_frame\_type()

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

ApiFrameType

## get\_frame\_type\_value()

Returns the frame type integer value of this packet.

**Returns** the frame type integer value of this packet.

Return type Integer

```
ApiFrameType
```

### is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

### op mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

#### output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

### to dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

## static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

**Return type** Bytearray

## static create\_packet (raw, operating\_mode)

Override method.

Returns SocketStatePacket.

### **Raises**

- InvalidPacketException if the bytearray length is less than 7. (start delim. + length (2 bytes) + frame type + socket ID + state + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. SOCKET\_STATUS.

• InvalidOperatingModeException - if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

socket_id
Returns the socket ID.
Returns the socket ID.
Return type Integer

state
Returns the socket state.
Returns The socket state.
Return type SocketState
See also:
```

## digi.xbee.packets.wifi module

SocketState

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a IO data sample RX indicator (Wi-Fi) packet. Packet is built using the parameters of the constructor or providing a valid API payload.

When the module receives an IO sample frame from a remote device, it sends the sample out the UART or SPI using this frame type. Only modules running API mode will be able to receive IO samples.

Among received data, some options can also be received indicating transmission parameters.

See also:

*XBeeAPIPacket* 

Class constructor. Instantiates a new IODataSampleRxIndicatorWifiPacket object with the provided parameters.

### **Parameters**

- src\_address (ipaddress.IPv4Address) the 64-bit source address.
- **rssi** (*Integer*) received signal strength indicator.
- **rx\_options** (*Integer*) bitfield indicating the receive options.
- rf\_data (Bytearray, optional) received RF data.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

Raises ValueError - if rf\_data is not None and it's not valid for create an IOSample.

#### See also:

```
IOSample
ipaddress.IPv4Address
ReceiveOptions
XBeeAPIPacket
```

### static create packet(raw, operating mode)

Override method.

Returns IODataSampleRxIndicatorWifiPacket.

### Raises

- InvalidPacketException if the bytearray length is less than 16. (start delim. + length (2 bytes) + frame type + source addr. (4 bytes) + rssi + receive options + rf data (5 bytes) + checksum = 16 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. IO\_DATA\_SAMPLE\_RX\_INDICATOR\_WIFI.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

# See also: XBeeAPIPacket.needs\_id() source\_address Returns the IPv4 address of the source device. **Returns** the IPv4 address of the source device. Return type ipaddress. IPv4Address See also: ipaddress.IPv4Address rssi Returns the received Signal Strength Indicator (RSSI). **Returns** the received Signal Strength Indicator (RSSI). Return type Integer receive\_options Returns the receive options bitfield. **Returns** the receive options bitfield. Return type Integer See also: ReceiveOptions rf data Returns the received RF data. **Returns** the received RF data. Return type Bytearray io sample Returns the IO sample corresponding to the data contained in the packet. **Returns** the IO sample of the packet, None if the packet has not any data or if the sample could not be generated correctly.

2.6. API reference 427

Return type IOSample

See also:

*IOSample* 

```
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
```

# is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

## Return type Boolean

#### op mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a remote AT command request (Wi-Fi) packet. Packet is built using the parameters of the constructor or providing a valid API payload.

Used to query or set module parameters on a remote device. For parameter changes on the remote device to take effect, changes must be applied, either by setting the apply changes options bit, or by sending an AC command to the remote node.

Remote command options are set as a bitfield.

If configured, command response is received as a RemoteATCommandResponseWifiPacket.

See also:

```
RemoteATCommandResponseWifiPacket
XBeeAPIPacket
```

Class constructor. Instantiates a new RemoteATCommandWifiPacket object with the provided parameters.

## **Parameters**

• frame id (integer) - the frame ID of the packet.

- dest\_address (ipaddress.IPv4Address) the IPv4 address of the destination device.
- tx\_options (Integer) bitfield of supported transmission options.
- command (String) AT command to send.
- parameter (Bytearray, optional) AT command parameter.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if length of *command* is different than 2.

#### See also:

```
ipaddress.IPv4Address
RemoteATCmdOptions
XBeeAPIPacket
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns RemoteATCommandWifiPacket

#### Raises

- InvalidPacketException if the Bytearray length is less than 17. (start delim. + length (2 bytes) + frame type + frame id + dest. addr. (8 bytes) + transmit options + command (2 bytes) + checksum = 17 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. REMOTE\_AT\_COMMAND\_REQUEST\_WIFI.
- InvalidOperatingModeException if operating\_mode is not supported.

# See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

## See also:

```
XBeeAPIPacket.needs id()
```

# dest\_address

Returns the IPv4 address of the destination device.

**Returns** the IPv4 address of the destination device.

Return type ipaddress. IPv4Address

See also:

ipaddress. IPv4Address

# transmit\_options

Returns the transmit options bitfield.

**Returns** the transmit options bitfield.

**Return type** Integer

See also:

RemoteATCmdOptions

## command

Returns the AT command.

Returns the AT command.

Return type String

# parameter

Returns the AT command parameter.

**Returns** the AT command parameter.

**Return type** Bytearray

# $frame\_id$

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

**Return type** Integer

# get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

**Returns** checksum value of this XBeePacket.

**Return type** Integer

See also:

factory

```
get_frame_spec_data()
     Override method.
     See also:
     XBeePacket.get frame spec data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is_broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op_mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to_dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
```

# **Return type** Dictionary

```
static unescape_data(data)
```

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a remote AT command response (Wi-Fi) packet. Packet is built using the parameters of the constructor or providing a valid API payload.

If a module receives a remote command response RF data frame in response to a Remote AT Command Request, the module will send a Remote AT Command Response message out the UART. Some commands may send back multiple frames for example, Node Discover (ND) command.

This packet is received in response of a RemoteATCommandPacket.

Response also includes an ATCommandStatus object with the status of the AT command.

# See also:

```
RemoteATCommandWifiPacket
ATCommandStatus
XBeeAPIPacket
```

Class constructor. Instantiates a new RemoteATCommandResponseWifiPacket object with the provided parameters.

## **Parameters**

- **frame\_id** (Integer) the frame ID of the packet.
- src\_address (ipaddress.IPv4Address) the IPv4 address of the source device.
- **command** (String) the AT command of the packet. Must be a string.
- resp\_status (ATCommandStatus) the status of the AT command.
- comm\_value (Bytearray, optional) the AT command response value.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## Raises

- ValueError if *frame\_id* is less than 0 or greater than 255.
- ValueError if length of *command* is different than 2.

# See also:

```
ATCommandStatus
ipaddress.IPv4Address
frame_id
    Returns the frame ID of the packet.
        Returns the frame ID of the packet.
        Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
    between the length field and the checksum field.
        Returns checksum value of this XBeePacket.
        Return type Integer
    See also:
    factory
get_frame_spec_data()
    Override method.
    See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
        Returns the frame type of this packet.
        Return type ApiFrameType
    See also:
    ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
        Returns the frame type integer value of this packet.
        Return type Integer
```

See also:

## ApiFrameType

# is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

# op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

## static create\_packet (raw, operating\_mode)

Override method.

**Returns** RemoteATCommandResponseWifiPacket.

#### Raises

- InvalidPacketException if the bytearray length is less than 17. (start delim. + length (2 bytes) + frame type + frame id + source addr. (8 bytes) + command (2 bytes) + receive options + checksum = 17 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. REMOTE\_AT\_COMMAND\_RESPONSE\_WIFI.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

# needs\_id()

Override method.

See also:

```
XBeeAPIPacket.needs_id()
```

## source\_address

Returns the IPv4 address of the source device.

**Returns** the IPv4 address of the source device.

Return type ipaddress. IPv4Address

See also:

```
ipaddress.IPv4Address
```

## command

Returns the AT command of the packet.

**Returns** the AT command of the packet.

Return type String

# status

Returns the AT command response status of the packet.

**Returns** the AT command response status of the packet.

Return type ATCommandStatus

See also:

ATCommandStatus

## command value

Returns the AT command response value.

**Returns** the AT command response value.

Return type Bytearray

# digi.xbee.packets.zigbee module

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Register Joining Device packet. Packet is built using the parameters of the constructor or providing a valid API payload.

Use this frame to securely register a joining device to a trust center. Registration is the process by which a node is authorized to join the network using a preconfigured link key or installation code that is conveyed to the trust center out-of-band (using a physical interface and not over-the-air).

If registering a device with a centralized trust center (EO = 2), then the key entry will only persist for KT seconds before expiring.

Registering devices in a distributed trust center (EO = 0) is persistent and the key entry will never expire unless explicitly removed.

To remove a key entry on a distributed trust center, this frame should be issued with a null (None) key. In a centralized trust center you cannot use this method to explicitly remove the key entries.

#### See also:

## *XBeeAPIPacket*

Class constructor. Instantiates a new RegisterJoiningDevicePacket object with the provided parameters.

# **Parameters**

- **frame\_id** (*integer*) the frame ID of the packet.
- registrant\_address (XBee64BitAddress) the 64-bit address of the destination device.
- options (RegisterKeyOptions) the register options indicating the key source.
- **key** (Bytearray) key of the device to register. Up to 16 bytes if entering a Link Key or up to 18 bytes (16-byte code + 2 byte CRC) if entering an Install Code.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

# See also:

XBee64BitAddress XBeeAPIPacket RegisterKeyOptions

```
static create_packet (raw, operating_mode)
```

Override method.

Returns Register Joining Device Packet.

#### Raises

- InvalidPacketException if the bytearray length is less than 17. (start delim. + length (2 bytes) + frame type + frame id + 64-bit registrant addr. (8 bytes) + 16-bit registrant addr. (2 bytes) + options + checksum = 17 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 2 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. REGISTER JOINING DEVICE.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

registrant_address
Returns the 64-bit registrant address.

Returns the 64-bit registrant address.
```

Return type XBee64BitAddress

XBee64BitAddress

# options

Returns the register options value.

**Returns** the register options indicating the key source.

Return type RegisterKeyOptions

See also:

See also:

```
RegisterKeyOptions
```

```
key
```

Returns the register key.

**Returns** the register key.

**Return type** Bytearray

# $frame_id$

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

# get\_checksum()

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

**Returns** checksum value of this XBeePacket.

Return type Integer

See also:

factory

# get\_frame\_spec\_data()

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

# get\_frame\_type()

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

*ApiFrameType* 

# ${\tt get\_frame\_type\_value}\:(\:)$

Returns the frame type integer value of this packet.

**Returns** the frame type integer value of this packet.

Return type Integer

See also:

```
ApiFrameType
```

# is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

**Return type** Boolean

# op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

**Returns** raw bytearray of the XBeePacket.

Return type Bytearray

## to dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Register Device Status packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame is sent out of the UART of the trust center as a response to a 0x24 Register Device frame, indicating whether the registration was successful or not.

See also:

```
RegisterJoiningDevicePacket
XBeeAPIPacket
```

Class constructor. Instantiates a new RegisterDeviceStatusPacket object with the provided parameters.

## **Parameters**

- **frame\_id** (*integer*) the frame ID of the packet.
- **status** (*ZigbeeRegisterStatus*) status of the register device operation.
- **op\_mode** (*OperatingMode*, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

**Raises** ValueError – if *frame\_id* is less than 0 or greater than 255.

## See also:

```
XBeeAPIPacket
ZigbeeRegisterStatus
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns RegisterDeviceStatusPacket.

## Raises

- InvalidPacketException if the bytearray length is less than 17. (start delim. + length (2 bytes) + frame type + frame id + status + checksum = 7 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 1 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. REGISTER\_JOINING\_DEVICE\_STATUS.
- InvalidOperatingModeException if operating\_mode is not supported.

#### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
See also:
```

# XBeeAPIPacket.needs\_id()

#### status

Returns the register device status.

**Returns** the register device status.

Return type ZigbeeRegisterStatus

```
See also:
     ZigbeeRegisterStatus
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get_checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
    See also:
     factory
get_frame_spec_data()
    Override method.
    See also:
     XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
    See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
```

ApiFrameType

#### is broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

Return type Boolean

#### op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters escaped** (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

## to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

**Return type** Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

class digi.xbee.packets.zigbee.RouteRecordIndicatorPacket(x64bit\_addr,

x16bit\_addr, rx\_opts, hops=None, op\_mode=<OperatingMode.API\_MODE: (1, 'API mode')>)

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Zigbee Route Record Indicator packet. Packet is built using the parameters of the constructor or providing a valid API payload.

The route record indicator is received whenever a device sends a Zigbee route record command. This is used with many-to-one routing to create source routes for devices in a network.

Among received data, some options can also be received indicating transmission parameters.

See also:

ReceiveOptions XBeeAPIPacket

Class constructor. Instantiates a new <code>RouteRecordIndicatorPacket</code> object with the provided parameters.

Parameters

- **x64bit\_addr** (*XBee64BitAddress*) The 64-bit source address.
- x16bit addr (XBee16BitAddress) The 16-bit source address.
- rx\_opts (Integer) Bitfield indicating the receive options.
- hops (List, optional, default=`None`) List of 16-bit address of intermediate hops in the source route (excluding source and destination).
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

## See also:

ReceiveOptions XBee16BitAddress XBee64BitAddress XBeeAPIPacket

# static create\_packet (raw, operating\_mode)

Override method.

Returns RouteRecordIndicatorPacket.

#### Raises

- InvalidPacketException If the bytearray length is less than 17. (start delim. + length (2 bytes) + frame type + 64bit addr. + 16bit addr. + Receive options + num of addrs + checksum = 17 bytes).
- InvalidPacketException If the length field of *raw* is different from its real length. (length field: bytes 1 and 3)
- InvalidPacketException If the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException If the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException If the frame type is not ApiFrameType. ROUTE\_RECORD\_INDICATOR.
- InvalidPacketException If the number of hops does not match with the number of 16-bit addresses.
- InvalidOperatingModeException If operating\_mode is not supported.

## See also:

See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

```
XBeeAPIPacket.needs_id()
is_broadcast()
    Override method.
    See also:
    XBeeAPIPacket.is_broadcast()
x64bit_source_addr
    Returns the 64-bit source address.
        Returns The 64-bit source address.
        Return type XBee64BitAddress
    See also:
    XBee64BitAddress
x16bit_source_addr
    Returns the 16-bit source address.
        Returns The 16-bit source address.
        Return type XBee16BitAddress
    See also:
    XBee16BitAddress
receive_options
    Returns the receive options bitfield.
        Returns The receive options bitfield.
        Return type Integer
    See also:
    ReceiveOptions
number_of_hops
    Returns the number of intermediate hops in the source route (excluding source and destination).
        Returns The number of addresses.
```

2.6. API reference 445

Return type Integer

## hops

Returns the list of intermediate hops starting from the closest to destination hop and finishing with the closest to the source (excluding source and destination).

**Returns** The list of 16-bit addresses of intermediate hops.

**Return type** List

See also:

XBee16BitAddress

# frame\_id

Returns the frame ID of the packet.

Returns the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

```
get_frame_spec_data()
```

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
```

Returns the frame type of this packet.

**Returns** the frame type of this packet.

Return type ApiFrameType

See also:

*ApiFrameType* 

```
get_frame_type_value()
```

Returns the frame type integer value of this packet.

**Returns** the frame type integer value of this packet.

```
Return type Integer
```

See also:

ApiFrameType

#### op mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

# output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

**Returns** dictionary with all info of the XBeePacket fields.

Return type Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters data** (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a Zigbee Create Source Route packet. This packet is built using the parameters of the constructor or providing a valid API payload.

This frame creates a source route in the node. A source route specifies the complete route a packet should travese to get from source to destination. Source routing should be used with many-to-one routing for best results.

Note: Both, 64-bit and 16-bit destination addresses are required when creating a source route. These are obtained when a Route Record Indicator (0xA1) frame is received.

See also:

```
RouteRecordIndicatorPacket
XBeeAPIPacket
```

Class constructor. Instantiates a new CreateSourceRoutePacket object with the provided parameters.

#### **Parameters**

- **frame\_id** (*integer*) the frame ID of the packet.
- x64bit\_addr (XBee64BitAddress) The 64-bit destination address.
- x16bit addr (XBee16BitAddress) The 16-bit destination address.
- route options (Integer) Route command options.
- hops (List, optional, default=`None`) List of 16-bit addresses of intermediate hops in the source route (excluding source and destination).
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

#### See also:

XBee16BitAddress XBee64BitAddress XBeeAPIPacket

static create\_packet (raw, operating\_mode)

Override method.

Returns CreateSourceRoutePacket.

#### Raises

- InvalidPacketException If the bytearray length is less than 18. (start delim. + length (2 bytes) + frame type + frame id + 64-bit addr. + 16-bit addr. + Route command options + num of addrs + hops 16-bit addrs + checksum = 18 bytes).
- InvalidPacketException If the length field of *raw* is different from its real length. (length field: bytes 1 and 3)
- InvalidPacketException If the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException If the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException If the frame type is not ApiFrameType. CREATE\_SOURCE\_ROUTE.
- InvalidPacketException If the number of hops does not match with the number of 16-bit addresses.
- InvalidOperatingModeException If operating\_mode is not supported.

# See also:

448

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
```

#### needs id()

Override method.

See also:

```
XBeeAPIPacket.needs id()
```

# x64bit\_dest\_addr

Returns the 64-bit destination address.

**Returns** The 64-bit destination address.

Return type XBee64BitAddress

See also:

XBee64BitAddress

# x16bit\_dest\_addr

Returns the 16-bit destination address.

**Returns** The 16-bit destination address.

Return type XBee16BitAddress

See also:

XBee16BitAddress

# route\_cmd\_options

Returns the route command options bitfield.

**Returns** The route command options bitfield.

**Return type** Integer

# number\_of\_hops

Returns the number of intermediate hops in the source route (excluding source and destination).

**Returns** The number of intermediate hops.

**Return type** Integer

# hops

Returns the list of intermediate hops starting from the closest to destination hop and finishing with the closest to the source (excluding source and destination).

**Returns** The list of 16-bit addresses of intermediate hops.

**Return type** List

See also:

XBeel6BitAddress

```
frame id
    Returns the frame ID of the packet.
         Returns the frame ID of the packet.
         Return type Integer
get checksum()
    Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes
     between the length field and the checksum field.
         Returns checksum value of this XBeePacket.
         Return type Integer
     See also:
     factory
get_frame_spec_data()
     Override method.
    See also:
    XBeePacket.get_frame_spec_data()
get_frame_type()
    Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
    Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
```

# is\_broadcast()

Returns whether this packet is broadcast or not.

**Returns** *True* if this packet is broadcast, *False* otherwise.

# Return type Boolean

#### op\_mode

Retrieves the operating mode in which this packet was read.

**Returns** The operating mode.

Return type OperatingMode

## output (escaped=False)

Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.

**Parameters** escaped (Boolean) – indicates if the raw bytearray must be escaped.

Returns raw bytearray of the XBeePacket.

Return type Bytearray

# to\_dict()

Returns a dictionary with all information of the XBeePacket fields.

Returns dictionary with all info of the XBeePacket fields.

Return type Dictionary

# static unescape\_data(data)

Un-escapes the provided bytearray data.

**Parameters** data (*Bytearray*) – the bytearray to unescape.

Returns data unescaped.

Return type Bytearray

class digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket (src\_address\_64,

updater\_address\_16,
rx\_options,
msg\_type,
block\_number,
target\_address\_64,
op\_mode=<OperatingMode.API\_MODE:
(1, 'API mode')>)

Bases: digi.xbee.packets.base.XBeeAPIPacket

This class represents a an Over The Air Firmware Update Status packet. Packet is built using the parameters of the constructor or providing a valid API payload.

This frame provides a status indication of a firmware update transmission.

If a query request returns a 0x15 (NACK) status, the target is likely waiting for a firmware update image. If no messages are sent to it for about 75 seconds, the target will timeout and accept new query messages.

If a query status returns a 0x51 (QUERY) status, then the target's bootloader is not active and will not respond to query messages.

See also:

EmberBootloaderMessageType
XBeeAPIPacket

Class constructor. Instantiates a new OTAFirmwareUpdateStatusPacket object with the provided parameters.

#### **Parameters**

- **src\_address\_64** (XBee64BitAddress) the 64-bit address of the device returning this answer.
- updater\_address\_16 (XBee16BitAddress) the 16-bit address of the updater device.
- **rx\_options** (*Integer*) bitfield indicating the receive options.
- msq\_type (EmberBootloaderMessageType) Ember bootloader message type
- block\_number (Integer) block number used in the update request.
- target\_address\_64 (XBee64BitAddress) the 64-bit address of the device that is being updated.
- **op\_mode** (OperatingMode, optional, default='OperatingMode.API\_MODE') The mode in which the frame was captured.

# See also:

```
XBeeAPIPacket
XBee16BitAddress
XBee64BitAddress
ReceiveOptions
EmberBootloaderMessageType
```

# frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

```
get_checksum()
```

Returns the checksum value of this XBeePacket. The checksum is the last 8 bits of the sum of the bytes between the length field and the checksum field.

Returns checksum value of this XBeePacket.

Return type Integer

See also:

```
factory
```

```
get_frame_spec_data()
```

Override method.

See also:

```
XBeePacket.get_frame_spec_data()
```

```
get_frame_type()
     Returns the frame type of this packet.
         Returns the frame type of this packet.
         Return type ApiFrameType
     See also:
     ApiFrameType
get_frame_type_value()
     Returns the frame type integer value of this packet.
         Returns the frame type integer value of this packet.
         Return type Integer
     See also:
     ApiFrameType
is broadcast()
     Returns whether this packet is broadcast or not.
         Returns True if this packet is broadcast, False otherwise.
         Return type Boolean
op mode
     Retrieves the operating mode in which this packet was read.
         Returns The operating mode.
         Return type OperatingMode
output (escaped=False)
     Returns the raw bytearray of this XBeePacket, ready to be send by the serial port.
         Parameters escaped (Boolean) – indicates if the raw bytearray must be escaped.
         Returns raw bytearray of the XBeePacket.
         Return type Bytearray
to dict()
     Returns a dictionary with all information of the XBeePacket fields.
         Returns dictionary with all info of the XBeePacket fields.
         Return type Dictionary
static unescape_data(data)
     Un-escapes the provided bytearray data.
         Parameters data (Bytearray) – the bytearray to unescape.
         Returns data unescaped.
         Return type Bytearray
```

# static create\_packet (raw, operating\_mode)

Override method.

Returns OTAFirmwareUpdateStatusPacket.

#### Raises

- InvalidPacketException if the bytearray length is less than 17. (start delim. + length (2 bytes) + frame type + source 64bit addr. (8 bytes) + updater 16bit addr. (2 bytes) + receive options + bootloader message type + block number + source 64bit addr. (8 bytes) + checksum = 27 bytes).
- InvalidPacketException if the length field of 'raw' is different from its real length. (length field: bytes 1 and 3)
- InvalidPacketException if the first byte of 'raw' is not the header byte. See SpecialByte.
- InvalidPacketException if the calculated checksum is different from the checksum field value (last byte).
- InvalidPacketException if the frame type is not ApiFrameType. OTA\_FIRMWARE\_UPDATE\_STATUS.
- InvalidOperatingModeException if operating\_mode is not supported.

## See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()

needs_id()
Override method.
See also:

XBeeAPIPacket.needs_id()

x64bit_source_addr
Returns the 64-bit source address.

Returns the 64-bit source address.

Return type XBee64BitAddress
```

# XBee64BitAddress

# x16bit\_updater\_addr

See also:

Returns the 16-bit updater address.

**Returns** the 16-bit updater address.

Return type XBee16BitAddress

## See also:

XBee16BitAddress

# receive\_options

Returns the receive options bitfield.

Returns the receive options bitfield.

Return type Integer

See also:

ReceiveOptions

# bootloader\_msg\_type

Returns the bootloader message type.

**Returns** the bootloader message type.

Return type EmberBootloaderMessageType

See also:

EmberBootloaderMessageType

# block\_number

Returns the block number of the request.

**Returns** the block number of the request.

Return type Integer

# x64bit\_target\_addr

Returns the 64-bit target address.

**Returns** the 64-bit target address.

Return type XBee64BitAddress

See also:

XBee64BitAddress

# digi.xbee.packets.factory module

This module provides functionality to build XBee packets from bytearray returning the appropriate XBeePacket subclass.

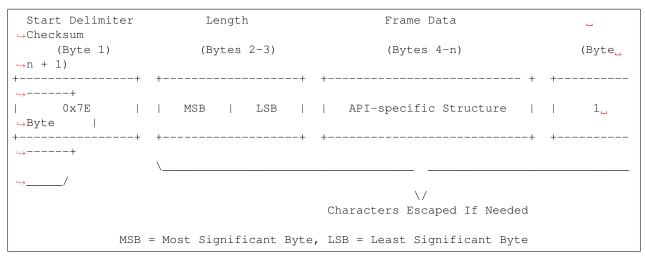
All the API and API2 logic is already included so all packet reads are independent of the XBee operating mode.

Two API modes are supported and both can be enabled using the AP (API Enable) command:

# API1 - API Without Escapes The data frame structure is defined as follows:

Start Delimiter  →Checksum	Length	Frame Data	_
(Byte 1)	(Bytes 2-3)	(Bytes 4-n)	(Byte_
<pre>→n + 1)</pre>			
++	+	+ ++	+
<b>⇔</b> +			
0x7E	MSB   LSB	API-specific Structure	1_
⇔Byte			
++	+	+ ++	+
<b>→</b> +			
MSB = Most Significant Byte, LSB = Least Significant Byte			

# API2 - API With Escapes The data frame structure is defined as follows:



When sending or receiving an API2 frame, specific data values must be escaped (flagged) so they do not interfere with the data frame sequencing. To escape an interfering data byte, the byte 0x7D is inserted before the byte to be escaped XOR'd with 0x20.

The data bytes that need to be escaped:

- 0x7E Frame Delimiter SpecialByte.
- 0x7D Escape
- 0x11 XON
- 0x13 XOFF

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

The frame data forms an API-specific structure as follows:

```
      Start Delimiter
      Length
      Frame Data

      Ghecksum
      (Byte 1)
      (Bytes 2-3)
      (Bytes 4-n)
      (Byte 2-3)

      Image: Start Delimiter
      Image: Start Delimite
```

(continues on next page)

(continued from previous page)

The cmdID frame (API-identifier) indicates which API messages will be contained in the cmdData frame (Identifier-specific data).

To unit\_test data integrity, a checksum is calculated and verified on non-escaped data.

## See also:

```
XBeePacket
OperatingMode
```

Creates a packet from raw data.

#### **Parameters**

- packet\_bytearray (Bytearray) the raw data of the packet to build.
- operating\_mode (OperatingMode) the operating mode in which the raw data has been captured.

# See also:

OperatingMode

# digi.xbee.util package

## **Submodules**

# digi.xbee.util.utils module

```
digi.xbee.util.utils.is_bit_enabled(number, position)

Returns whether the bit located at position within number is enabled.
```

## **Parameters**

- number (Integer) the number to check if a bit is enabled.
- **position** (*Integer*) the position of the bit to check if is enabled in *number*.

## Returns

True if the bit located at position within number is enabled, False otherwise.

# Return type Boolean

```
digi.xbee.util.utils.get_int_from_byte(number, offset, length)
```

Reads an integer value from the given byte using the provived bit offset and length.

#### **Parameters**

- **number** (*Integer*) Byte to read the integer from.
- **offset** (*Integer*) Bit offset inside the byte to start reading (LSB = 0, MSB = 7).
- **length** (*Integer*) Number of bits to read.

Returns The integer value read.

# Return type Integer

**Raises** ValueError – If *number is lower than 0 or higher than 255. If 'offset* is lower than 0 or higher than 7. If *length* is lower than 0 or higher than 8. If *offset* + *length* is higher than 8.

```
digi.xbee.util.utils.hex_string_to_bytes(hex_string)
```

Converts a String (composed by hex. digits) into a bytearray with same digits.

Parameters hex\_string (String) - String (made by hex. digits) with "0x" header or not.

**Returns** bytearray containing the numeric value of the hexadecimal digits.

# **Return type** Bytearray

Raises ValueError – if invalid literal for int() with base 16 is provided.

# **Example**

```
>>> a = "0xFFFE"
>>> for i in hex_string_to_bytes(a): print(i)
255
254
>>> print(type(hex_string_to_bytes(a)))
<type 'bytearray'>
```

```
>>> b = "FFFE"
>>> for i in hex_string_to_bytes(b): print(i)
255
254
>>> print(type(hex_string_to_bytes(b)))
<type 'bytearray'>
```

## digi.xbee.util.utils.int\_to\_bytes(number, num\_bytes=None)

Converts the provided integer into a bytearray.

If number has less bytes than  $num\_bytes$ , the resultant bytearray is filled with zeros (0x00) starting at the beginning.

If *number* has more bytes than *num\_bytes*, the resultant bytearray is returned without changes.

## **Parameters**

- **number** (*Integer*) the number to convert to a bytearray.
- num\_bytes (Integer) the number of bytes that the resultant bytearray will have.

**Returns** the bytearray corresponding to the provided number.

Return type Bytearray

# **Example**

```
>>> a=0xFFFE
>>> print([i for i in int_to_bytes(a)])
[255,254]
>>> print(type(int_to_bytes(a)))
<type 'bytearray'>
```

```
digi.xbee.util.utils.length_to_int(byte_array)
```

Calculates the length value for the given length field of a packet. Length field are bytes 1 and 2 of any packet.

**Parameters** byte\_array (Bytearray) - length field of a packet.

**Returns** the length value.

Return type Integer

**Raises** ValueError – if byte\_array is not a valid length field (it has length distinct than 0).

# **Example**

```
>>> b = bytearray([13,14])
>>> c = length_to_int(b)
>>> print("0x%02X" % c)
0x1314
>>> print(c)
4884
```

```
digi.xbee.util.utils.bytes_to_int(byte_array)
```

Converts the provided bytearray in an Integer. This integer is result of concatenate all components of *byte\_array* and convert that hex number to a decimal number.

**Parameters** byte\_array (Bytearray) – bytearray to convert in integer.

**Returns** the integer corresponding to the provided bytearray.

Return type Integer

# **Example**

```
>>> x = bytearray([0xA,0x0A,0x0A]) #this is 0xA0A0A

>>> print(bytes_to_int(x))

657930

>>> b = bytearray([0x0A,0xAA]) #this is 0xAAA

>>> print(bytes_to_int(b))

2730
```

```
digi.xbee.util.utils.ascii_to_int(array)
```

Converts a bytearray containing the ASCII code of each number digit in an Integer. This integer is result of the number formed by all ASCII codes of the bytearray.

**Parameters array** (*Bytearray*) – bytearray to convert in integer.

# **Example**

```
>>> x = bytearray( [0x31,0x30,0x30] ) #0x31 => ASCII code for number 1.
#0x31,0x30,0x30 <==> 1,0,0
>>> print(ascii_to_int(x))
100
```

```
digi.xbee.util.utils.int_to_ascii(number)
```

Converts an integer number to a bytearray. Each element of the bytearray is the ASCII code that corresponds to the digit of its position.

**Parameters** number (Integer) – the number to convert to an ASCII bytearray.

**Returns** the bytearray containing the ASCII value of each digit of the number.

Return type Bytearray

# **Example**

```
>>> x = int_to_ascii(100)

>>> print(x)

100

>>> print([i for i in x])

[49, 48, 48]
```

```
digi.xbee.util.utils.int_to_length(number)
```

Converts an integer into a bytearray of 2 bytes corresponding to the length field of a packet. If this bytearray has length 1, a byte with value 0 is added at the beginning.

**Parameters** number (Integer) – the number to convert to a length field.

**Returns** The bytearray.

**Return type** Bytearray

**Raises** ValueError – if *number* is less than 0 or greater than 0xFFFF.

# **Example**

```
>>> a = 0
>>> print(hex_to_string(int_to_length(a)))
00 00
```

```
>>> a = 8
>>> print(hex_to_string(int_to_length(a)))
00 08
```

```
>>> a = 200
>>> print(hex_to_string(int_to_length(a)))
00 C8
```

```
>>> a = 0xFF00
>>> print(hex_to_string(int_to_length(a)))
FF 00
```

```
>>> a = 0xFF
>>> print(hex_to_string(int_to_length(a)))
00 FF
```

```
digi.xbee.util.utils.hex_to_string(byte_array, pretty=True)
```

Returns the provided bytearray in a pretty string format. All bytes are separated by blank spaces and printed in hex format.

#### **Parameters**

- **byte\_array** (*Bytearray*) the bytearray to print in pretty string.
- **pretty** (Boolean, optional) *True* for pretty string format, *False* for plain string format. Default to *True*.

**Returns** the bytearray formatted in a string format.

# Return type String

digi.xbee.util.utils.doc\_enum(enum\_class, descriptions=None)

Returns a string with the description of each value of an enumeration.

#### **Parameters**

- enum\_class (Enumeration) the Enumeration to get its values documentation.
- **descriptions** (*dictionary*) each enumeration's item description. The key is the enumeration element name and the value is the description.

**Returns** the string listing all the enumeration values and their descriptions.

## **Return type** String

```
\verb|digi.xbee.util.utils.enable_logger| (name, level=10)
```

Enables a logger with the given name and level.

# **Parameters**

- name (String) name of the logger to enable.
- **level** (*Integer*) logging level value.

Assigns a default formatter and a default handler (for console).

```
digi.xbee.util.utils.disable_logger(name)
```

Disables the logger with the give name.

**Parameters** name (String) – the name of the logger to disable.

```
digi.xbee.util.utils.deprecated(version, details='None')
```

Decorates a method to mark as deprecated. This adds a deprecation note to the method docstring and also raises a warning. DeprecationWarning.

# **Parameters**

- **version** (*String*) Version that deprecates this feature.
- **details** (String, optional, default=`None`) Extra details to be added to the method docstring and warning.

# digi.xbee.util.xmodem module

```
exception digi.xbee.util.xmodem.XModemException
    Bases: Exception
```

This exception will be thrown when any problem related with the XModem/YModem transfer occurs.

All functionality of this class is the inherited from Exception.

# with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

## exception digi.xbee.util.xmodem.XModemCancelException

Bases: digi.xbee.util.xmodem.XModemException

This exception will be thrown when the XModem/YModem transfer is cancelled by the remote end.

All functionality of this class is the inherited from Exception.

#### with traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

Sends a file using the XModem protocol to a remote end.

## **Parameters**

- **src\_path** (*String*) absolute path of the file to transfer.
- write\_cb (Function) function to execute in order to write data to the remote end. Takes the following arguments:
  - The data to write as byte array.

The function returns the following:

Boolean: *True* if the write succeeded. *False* otherwise.

- **read\_cb** (Function) function to execute in order to read data from the remote end. Takes the following arguments:
  - The size of the data to read.
  - The timeout to wait for data. (seconds)

The function returns the following:

Bytearray: the read data, None if data could not be read

- **progress\_cb** (Function, optional) function to execute in order to receive progress information. Takes the following arguments:
  - The progress percentage as integer.
- log (Logger, optional) logger used to log transfer debug messages

## Raises

- ValueError if any input value is not valid.
- XModemCancelException if the transfer is cancelled by the remote end.
- XModemException if there is any error during the file transfer.

Sends a file using the YModem protocol to a remote end.

#### **Parameters**

• **src\_path** (*String*) – absolute path of the file to transfer.

- write\_cb (Function) function to execute in order to write data to the remote end. Takes the following arguments:
  - The data to write as byte array.

The function returns the following:

Boolean: True if the write succeeded, False otherwise

- **read\_cb** (Function) function to execute in order to read data from the remote end. Takes the following arguments:
  - The size of the data to read.
  - The timeout to wait for data. (seconds)

The function returns the following:

Bytearray: the read data, None if data could not be read

- **progress\_cb** (Function, optional) function to execute in order to receive progress information. Takes the following arguments:
  - The progress percentage as integer.
- log (Logger, optional) logger used to log transfer debug messages

#### **Raises**

- ValueError if any input value is not valid.
- XModemCancelException if the transfer is cancelled by the remote end.
- XModemException if there is any error during the file transfer.

Retrieves a file using the YModem protocol from a remote end.

## **Parameters**

- **dest\_path** (String) absolute path to store downloaded file in.
- write\_cb (Function) function to execute in order to write data to the remote end. Takes the following arguments:
  - The data to write as byte array.

The function returns the following:

Boolean: True if the write succeeded, False otherwise

- **read\_cb** (Function) function to execute in order to read data from the remote end. Takes the following arguments:
  - The size of the data to read.
  - The timeout to wait for data. (seconds)

The function returns the following:

Bytearray: the read data, None if data could not be read

- **crc** (Boolean, optional) *True* to use 16-bit CRC verification, *False* for standard 1 byte checksum. Defaults to *True*.
- **progress\_cb** (Function, optional) function to execute in order to receive progress information. Takes the following arguments:

- The progress percentage as integer.
- log (Logger, optional) logger used to log download debug messages

#### Raises

- ValueError if any input value is not valid.
- XModemCancelException if the file download is cancelled by the remote end.
- XModemException if there is any error during the file download process.

#### **Submodules**

## digi.xbee.comm interface module

```
class digi.xbee.comm_interface.XBeeCommunicationInterface
    Bases: object
```

This class represents the way the communication with the local XBee is established.

## open()

Establishes the underlying hardware communication interface.

Subclasses may throw specific exceptions to signal implementation specific errors.

#### close()

Terminates the underlying hardware communication interface.

Subclasses may throw specific exceptions to signal implementation specific hardware errors.

## is\_interface\_open

Returns whether the underlying hardware communication interface is active or not.

**Returns** *True* if the interface is active, *False* otherwise.

**Return type** Boolean

## wait\_for\_frame (operating\_mode)

Reads the next API frame packet.

## This method blocks until:

- · A complete frame is read, in which case returns it.
- The configured timeout goes by, in which case returns *None*.
- Another thread calls quit\_reading, in which case returns *None*.

This method is not thread-safe, so no more than one thread should invoke it at the same time.

Subclasses may throw specific exceptions to signal implementation specific hardware errors.

**Parameters operating\_mode** (OperatingMode) – The operating mode of the XBee connected to this hardware interface. Note: If this parameter does not match the connected XBee configuration, the behavior is undefined.

## Returns

The read packet as bytearray if a packet is read, *None* otherwise.

Return type Bytearray

## quit\_reading()

Makes the thread (if any) blocking on wait\_for\_frame return.

If a thread was blocked on wait\_for\_frame, this method blocks (for a maximum of 'timeout' seconds) until the blocked thread is resumed.

## write\_frame (frame)

Writes an XBee frame to the underlying hardware interface.

Subclasses may throw specific exceptions to signal implementation specific hardware errors.

**Parameters frame** (*Bytearray*) – The XBee API frame packet to write. If the bytearray does not correctly represent an XBee frame, the behaviour is undefined.

#### get\_network (local\_xbee)

Returns the XBeeNetwork object associated to the XBeeDevice associated to this XBeeCommunicationInterface.

Some XBeeCommunicationInterface implementations may need to handle the 'XBeeNetwork associated to the XBeeDevice themselves. If that is the case, a implementation-specific XBeeNetwork object that complains to the generic XBeeNetwork class will be returned. Otherwise, this method returns None and the associated XBeeNetwork is handled as for a serial-connected XBeeDevice.

**Parameters** local\_xbee (XBeeDevice) - The local XBee device.

#### Returns

**class:** .XBeeNetwork: None if the XBeeNetwork should handled as usual, otherwise a XBeeNetwork object.

## get\_local\_xbee\_info()

Returns a tuple with the local XBee information.

This is used when opening the local XBee. If this information is provided, it is used as internal XBee data, if not provided, the data is requested to the XBee.

#### Returns

**Tuple with local XBee information: operation mode (int),** hardware version (int), firmware version (int), 64-bit address (string), 16-bit address (string), node identifier (string), and role (int).

Return type Tuple

## supports\_update\_firmware()

Returns if the interface supports the firmware update feature.

**Returns** *True* if it is supported, *False* otherwise.

**Return type** Boolean

Performs a firmware update operation of the provided XBee.

## **Parameters**

- xbee (AbstractXBeeDevice) Local or remote XBee node to be updated.
- xml\_fw\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_fw\_file** (String, optional) Location of the XBee binary firmware file.
- bootloader\_fw\_file (String, optional) Location of the bootloader binary firmware file.

- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the update process.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the local XBee is not open.
- InvalidOperatingModeException If the local XBee operating mode is invalid.
- OperationNotSupportedException If the firmware update is not supported in the XBee.
- Firmware Update Exception If there is any error performing the firmware update.

## supports\_apply\_profile()

Returns if the interface supports the apply profile feature.

**Returns** *True* if it is supported, *False* otherwise.

Return type Boolean

apply\_profile (xbee, profile\_path, timeout=None, progress\_callback=None)

Applies the given XBee profile to the XBee device.

#### **Parameters**

- **xbee** (AbstractXBeeDevice) Local or remote XBee node to be updated.
- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the apply profile.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

#### Raises

- XBeeException If the local XBee is not open.
- InvalidOperatingModeException If the local XBee operating mode is invalid.
- UpdateProfileException If there is any error applying the XBee profile.
- $\bullet$  OperationNotSupportedException If XBee profiles are not supported in the XBee.

#### timeout

Returns the read timeout.

**Returns** Read timeout in seconds.

Return type Integer

## digi.xbee.devices module

Bases: object

This class provides common functionality for all XBee devices.

Class constructor. Instantiates a new AbstractXBeeDevice object with the provided parameters.

#### **Parameters**

- local\_xbee\_device (XBeeDevice, optional, default='None') Only necessary if XBee is remote. The local XBee to be the connection interface to communicate with the remote XBee one.
- **serial\_port** (*XBeeSerialPort*, optional, default='None') Only necessary if the XBee device is local. The serial port to communicate with this XBee.
- (Integer, optional, default (sync\_ops\_timeout) AbstractXBeeDevice.\_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS): Timeout (in seconds) for all synchronous operations.
- **comm\_iface** (XBeeCommunicationInterface, optional, default='None') Only necessary if the XBee is local. The hardware interface to communicate with this XBee.

#### See also:

XBeeDevice XBeeSerialPort

## update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters device** (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

get\_parameter (parameter, parameter\_value=None, apply=None)
Returns the value of the provided parameter via an AT Command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): Parameter to get.
- parameter\_value (Bytearray, optional, default=`None`) Value of the parameter to execute (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

Returns Parameter value.

Return type Bytearray

## Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.execute_command()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
```

## set\_parameter (parameter, value, apply=None)

Sets the value of a parameter via an AT Command.

Any parameter changes are applied automatically, if *apply* is *True* or if it is *None* and apply flag is enabled (*is\_apply\_changes\_enabled()*)

You can set this flag via the method AbstractXBeeDevice.enable\_apply\_changes().

This only applies modified values in the XBee configuration, to save changed parameters permanently (between resets), use <code>AbstractXBeeDevice.write\_changes()</code>.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): Parameter to set.
- **value** (*Bytearray*) Value of the parameter.
- apply (Boolean, optional, default=`None`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If parameter is None or value is None.

## See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.execute_command()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()
```

execute\_command(parameter, value=None, apply=None)

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- value (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()
```

## apply\_changes()

Applies changes via 'AC' command.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code> to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

#### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### reset()

Performs a software reset on this XBee and blocks until the process is completed.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## read\_device\_info(init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

#### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if True.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.is device info complete()
```

## determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

## Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

# is device info complete() Returns whether XBee node information is complete. Return type Boolean

**Returns** *True* if node information is complete, *False* otherwise.

See also:

```
AbstractXBeeDevice.read_device_info()
```

#### get\_node\_id()

Returns the node identifier ('NI') value of the XBee.

Returns Node identifier ('NI') of the XBee.

Return type String

#### set node id (node id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

#### Raises

- ValueError If *node id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_hardware\_version()

Returns the hardware version of the XBee.

Returns Hardware version of the XBee.

Return type Hardware Version

See also:

**HardwareVersion** 

## get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

**Return type** Bytearray

## get\_protocol()

Returns the current protocol of the XBee.

**Returns** Current protocol of the XBee.

Return type XBeeProtocol

```
See also:
    XBeeProtocol
get_16bit_addr()
    Returns the 16-bit address of the XBee.
        Returns 16-bit address of the XBee.
        Return type XBee16BitAddress
    See also:
    XBee16BitAddress
set 16bit addr(value)
    Sets the 16-bit address of the XBee.
        Parameters value (XBee16BitAddress) – New 16-bit address of the XBee.
        Raises
             • TimeoutException – If response is not received before the read timeout expires.
             • XBeeException – If the XBee's communication interface is closed.
             • InvalidOperatingModeException - If the XBee's operating mode is not API or
              ESCAPED API. This method only checks the cached value of the operating mode.
             • ATCommandException – If response is not as expected.
             • OperationNotSupportedException - If the protocol is not 802.15.4.
get_64bit_addr()
    Returns the 64-bit address of the XBee.
        Returns 64-bit address of the XBee.
        Return type XBee64BitAddress
    See also:
    XBee64BitAddress
get_role()
    Gets the XBee role.
        Returns the role of the XBee.
        Return type Role
    See also:
```

Role

#### get\_current\_frame\_id()

Returns the last used frame ID.

Returns Last used frame ID.

Return type Integer

#### enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

## is\_apply\_changes\_enabled()

Returns whether apply changes flag is enabled.

**Returns** *True* if apply changes flag is enabled, *False* otherwise.

Return type Boolean

## is\_remote()

Determines whether XBee is remote.

**Returns** *True* if the XBee is remote, *False* otherwise.

**Return type** Boolean

## set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

## get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

## get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

Returns 64-bit address of destination XBee.

Return type XBee64BitAddress

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
XBee64BitAddress
set dest address()
```

#### set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

#### See also:

```
XBee64BitAddress
get_dest_address()
```

#### get\_pan\_id()

Returns the operating PAN ID of the XBee.

Returns Operating PAN ID of the XBee.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
set_pan_id()
```

## set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

474

```
get_pan_id()
```

## get\_power\_level()

Returns the power level of the XBee.

**Returns** Power level of the XBee.

Return type PowerLevel

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
PowerLevel
set_power_level()
```

## set\_power\_level (power\_level)

Sets the power level of the XBee.

Parameters power\_level (PowerLevel) - New power level of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
PowerLevel
get_power_level()
```

## set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

#### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

## Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
IOLine
IOMode
get_io_configuration()
```

## get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (*IOLine*) – IO line to get its configuration.

**Returns** IO mode of the IO line provided.

Return type IOMode

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
IOLine
IOMode
set_io_configuration()
```

## get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set_io_sampling_rate()
```

## set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
get_io_sampling_rate()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

Returns IO sample read from the XBee.

Return type IOSample

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOSample* 

## get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC</code>.

**Parameters** io\_line (*IOLine*) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

#### See also:

```
IOLine
set_io_configuration()
```

## set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

#### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or cycle is not between 0 and 100.

## See also:

```
IOLine
IOMode.PWM
```

## get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

**Returns** PWM duty cycle of the given IO line.

#### Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

#### See also:

*IOLine* 

#### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

#### See also:

```
IOLine
IOValue
set io configuration()
```

## set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

## **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

• ATCommandException – If response is not as expected.

#### See also:

*IOLine IOValue* 

## set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOLine* 

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

Returns API output mode of the XBee.

Return type APIOutputMode

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*APIOutputMode* 

#### get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

## set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

**Parameters** api\_output\_mode (APIOutputMode) - New API output mode.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

*APIOutputMode* 

## set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

*APIOutputModeBit* 

#### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method AbstractXBeeDevice.update\_bluetooth\_password().

Note that your XBee must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### disable bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- $\bullet \ \, \hbox{\tt TimeoutException-If response is not received before the read timeout expires.}$
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

Return type String

#### Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

Performs a firmware update operation of the XBee.

#### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet \ \, {\tt OperationNotSupportedException-If} \ \, \textbf{XBee does not support firmware update}.$
- FirmwareUpdateException If there is any error during the firmware update.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

## **Parameters**

• **profile\_path** (*String*) – Path of the XBee profile file to apply.

- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

#### get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

 $\textbf{Raises} \ \texttt{FileSystemNotSupportedException} - If \ \textbf{the XBeedoes not support filesystem}.$ 

## reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

## scan\_counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

#### log

Returns the XBee logger.

Returns The XBee device logger.

Return type Logger

```
\verb"class" digi.xbee.devices.XBeeDevice" (\textit{port=None},
```

baud\_rate=None,

data\_bits=<sphinx.ext.autodoc.importer.\_MockObject object>, stop\_bits=<sphinx.ext.autodoc.importer.\_MockObject object>, parity=<sphinx.ext.autodoc.importer.\_MockObject object>, flow\_control=<FlowControl.NONE: None>, \_sync\_ops\_timeout=4, comm\_iface=None)

Bases: digi.xbee.devices.AbstractXBeeDevice

This class represents a non-remote generic XBee.

This class has fields that are events. Its recommended to use only the append() and remove() method on them, or -= and += operators. If you do something more with them, it's for your own risk.

Class constructor. Instantiates a new XBeeDevice with the provided parameters.

## **Parameters**

- **port** (*String*) Serial port identifier. Depends on operating system. e.g. '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
- baud\_rate (Integer, optional, default=`None`) Serial port baud rate.
- (Integer, default (\_sync\_ops\_timeout) serial.EIGHTBITS): Port bitsize.
- (Integer, default serial.STOPBITS\_ONE): Port stop bits.
- (Character, default (parity) serial.PARITY\_NONE): Port parity.
- (Integer, default FlowControl.NONE): Port flow control.
- (Integer, default 4): Read timeout (in seconds).
- comm\_iface (XBeeCommunicationInterface) Communication interface.

Raises All exceptions raised by PySerial's Serial class constructor.

#### See also:

PySerial documentation: http://pyserial.sourceforge.net

## TIMEOUT\_READ\_PACKET = 3

Timeout to read packets.

## classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

#### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

Returns XBee object created.

Return type XBeeDevice

**Raises** SerialException – If the port to open does not exist or is already opened.

See also:

XBeeDevice

## open (force\_settings=False)

Opens the communication with the XBee and loads information about it.

<sup>&</sup>quot;port" -> Port number.

<sup>&</sup>quot;bitSize" -> Bit size.

<sup>&</sup>quot;stopBits" -> Stop bits.

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

<sup>&</sup>quot;timeout" for -> Timeout for synchronous operations (in seconds).

**Parameters force\_settings** (Boolean, optional, default=`False`) – *True* to open the device ensuring/forcing that the specified serial settings are applied even if the current configuration is different, *False* to open the device with the current configuration.

## **Raises**

- TimeoutException If there is any problem with the communication.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee is already opened.

## close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

#### serial\_port

Returns the serial port associated to the XBee, if any.

#### Returns

**Serial port of the XBee.** *None* **if the** local XBee does not use serial communication.

Return type XBeeSerialPort

See also:

XBeeSerialPort

## comm\_iface

Returns the hardware interface associated to the XBee.

**Returns** Hardware interface of the XBee.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

## operating\_mode

Returns the operating mode of this XBee.

**Returns** OperatingMode. This XBee operating mode.

get\_parameter (parameter, parameter\_value=None, apply=None)
Override.

See also:

```
AbstractXBeeDevice.get_parameter()
```

 $\verb|set_parameter| (parameter, value, apply=None)|$ 

Override.

**See:** AbstractXBeeDevice.set parameter()

## send\_data (remote\_xbee, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee synchronously.

This method will wait for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### **Returns** The response.

## Return type XBeePacket

#### Raises

- ValueError If remote\_xbee is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

#### See also:

RemoteXBeeDevice XBeePacket

## send\_data\_async (remote\_xbee, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee.

This method does not wait for a response.

## **Parameters**

- remote\_xbee (RemoteXBeeDevice) the remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- ValueError If remote\_xbee is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

#### See also:

RemoteXBeeDevice

## send\_data\_broadcast (data, transmit\_options=0)

Sends the provided data to all the XBee nodes of the network (broadcast).

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

The received timeout is configured using method AbstractXBeeDevice. set\_sync\_ops\_timeout() and can be consulted with AbstractXBeeDevice. get\_sync\_ops\_timeout() method.

#### **Parameters**

- data (String or Bytearray) Data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

#### send user data relay (local interface, data)

Sends the given data to the given XBee local interface.

## **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

#### See also:

XBeeLocalInterface

## send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

## Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

## send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send user data relay()
```

## read\_data(timeout=None)

Reads new data received by this XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

**Parameters timeout** (*Integer*, *optional*) – Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

#### Returns

Read message or None if this XBee did not receive new data.

Return type XBeeMessage

## Raises

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

## See also:

XBeeMessage

#### read\_data\_from (remote\_xbee, timeout=None)

Reads new data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

## **Parameters**

- $remote\_xbee$  (RemoteXBeeDevice) Remote XBee that sent the data.
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

#### Returns

**Read message sent by** remote\_xbee or None if this XBee did not receive new data.

Return type XBeeMessage

#### **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

#### See also:

```
XBeeMessage
RemoteXBeeDevice
```

## has\_packets()

Returns if there are pending packets to read. This does not include explicit packets.

Returns True if there are pending packets, False otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_explicit_packets()
```

## has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_packets()
```

#### flush\_queues()

Flushes the packets queue.

## reset()

Override method.

See also:

AbstractXBeeDevice.reset()

#### add\_packet\_received\_callback(callback)

Adds a callback for the event PacketReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

## add\_data\_received\_callback (callback)

Adds a callback for the event DataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The data received as an XBeeMessage.

## add\_modem\_status\_received\_callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

## add\_io\_sample\_received\_callback(callback)

Adds a callback for the event IOSampleReceived.

Parameters callback (Function) – The callback. Receives three arguments.

- The received IO sample as an IOSample.
- The remote XBee which sent the packet as a RemoteXBeeDevice.
- The time in which the packet was received as an Integer.

## add expl data received callback(callback)

Adds a callback for the event ExplicitDataReceived.

**Parameters** callback (Function) - The callback. Receives one argument.

• The explicit data received as a *ExplicitXBeeMessage*.

#### add\_user\_data\_relay\_received\_callback (callback)

Adds a callback for the event RelayDataReceived.

Parameters callback (Function) - The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

#### add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

#### add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

#### add\_socket\_state\_received\_callback (callback)

Adds a callback for the event SocketStateReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a *SocketState*.

## add\_socket\_data\_received\_callback(callback)

Adds a callback for the event SocketDataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

## add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

## add\_fs\_frame\_received\_callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

## See also:

AbstractXBeeDevice FSCmd ReceiveOptions

## del\_packet\_received\_callback(callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_data\_received\_callback (callback)

Deletes a callback for the callback list of <code>DataReceived</code> event.

**Parameters** callback (Function) – The callback to delete.

#### del\_modem\_status\_received\_callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

**Parameters** callback (Function) – The callback to delete.

#### del\_io\_sample\_received\_callback (callback)

Deletes a callback for the callback list of IOSampleReceived event.

**Parameters** callback (Function) – The callback to delete.

#### del\_expl\_data\_received\_callback (callback)

Deletes a callback for the callback list of ExplicitDataReceived event.

**Parameters** callback (Function) – The callback to delete.

#### del\_user\_data\_relay\_received\_callback(callback)

Deletes a callback for the callback list of RelayDataReceived event.

Parameters callback (Function) – The callback to delete.

## del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of SocketStateReceived event.

Parameters callback (Function) - The callback to delete.

## del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

#### del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

## ${\tt del\_fs\_frame\_received\_callback}\ (callback)$

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

#### get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

## is\_open()

Returns whether this XBee is open.

**Returns** Boolean. *True* if this XBee is open, *False* otherwise.

#### is remote()

Override method.

#### See also:

```
AbstractXBeeDevice.is remote()
```

## get\_network()

Returns the network of this XBee.

**Returns** The XBee network.

Return type XBeeNetwork

## send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

#### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

**Returns** Received response packet.

Return type XBeePacket

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

#### See also:

XBeePacket

## send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

#### **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

#### Returns

494

**Response packet if sync is True, None** otherwise.

Return type XBeePacket

#### Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

#### See also:

XBeePacket

## get\_next\_frame\_id()

Returns the next frame ID of the XBee.

**Returns** The next frame ID of the XBee.

Return type Integer

## add\_route\_received\_callback (callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

Parameters callback (Function) - The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

## See also:

```
XBeeDevice.del_route_received_callback()
```

## del\_route\_received\_callback (callback)

Deletes a callback for the callback list of RouteReceived event.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeDevice.add route received callback()
```

## get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

## For Zigbee:

• 'AR' parameter of the local node must be configured with a value different from 'FF'.

• Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

## **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

#### **Returns**

#### **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

## Return type Tuple

#### apply\_changes()

Applies changes via 'AC' command.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

#### **Parameters**

- **profile\_path** (String) Path of the XBee profile file to apply.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

## Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

• UpdateProfileException - If there is any error applying the XBee profile.

#### determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (Bytearray) Firmware version to get its protocol.

#### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

## disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

## enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method <code>AbstractXBeeDevice.update\_bluetooth\_password()</code>.

Note that your XBee must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## $\verb"execute_command" (parameter, value=None, apply=None)$

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- value (bytearray, optional, default=`None`) Command value (if any).

• apply (Boolean, optional, default=`None`) - True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()

get_16bit_addr()
Returns the 16-bit address of the XBee.
Return type XBee16BitAddress
See also:

XBee16BitAddress

get_64bit_addr()
Returns the 64-bit address of the XBee.
Returns 64-bit address of the XBee.
```

## XBee64BitAddress

get\_adc\_value(io\_line)

See also:

Returns the analog value of the provided IO line.

Return type XBee64BitAddress

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC</code>.

Parameters io\_line (IOLine) - IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

## Return type Integer

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
set_io_configuration()
```

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*APIOutputMode* 

# ${\tt get\_api\_output\_mode\_value}\;(\;)$

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

**Raises** 

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

digi.xbee.models.mode.APIOutputModeBit

# get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

Return type String

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_current\_frame\_id()

Returns the last used frame ID.

Returns Last used frame ID.

Return type Integer

# get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

**Returns** 64-bit address of destination XBee.

Return type XBee64BitAddress

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
XBee64BitAddress
set_dest_address()
```

## get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

## See also:

```
IOLine
IOValue
set_io_configuration()
```

## get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

## get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

Return type Bytearray

## get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

HardwareVersion

## get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

**Returns** IO mode of the IO line provided.

Return type IOMode

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
IOLine
IOMode
set_io_configuration()
```

## get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set_io_sampling_rate()
```

## get\_node\_id()

Returns the node identifier ('NI') value of the XBee.

Returns Node identifier ('NI') of the XBee.

Return type String

## get\_pan\_id()

Returns the operating PAN ID of the XBee.

Returns Operating PAN ID of the XBee.

Return type Bytearray

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set pan id()
```

## get\_power\_level()

Returns the power level of the XBee.

**Returns** Power level of the XBee.

Return type PowerLevel

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
PowerLevel
set_power_level()
```

# get\_protocol()

Returns the current protocol of the XBee.

**Returns** Current protocol of the XBee.

Return type XBeeProtocol

See also:

XBeeProtocol

# get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

Parameters io\_line (IOLine) - IO line to get its PWM duty cycle.

**Returns** PWM duty cycle of the given IO line.

Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If *io\_line* has no PWM capability.

## See also:

```
IOLine
```

```
get_role()
```

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

# get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

# is\_apply\_changes\_enabled()

Returns whether apply changes flag is enabled.

**Returns** *True* if apply changes flag is enabled, *False* otherwise.

Return type Boolean

# $\verb|is_device_info_complete|()$

Returns whether XBee node information is complete.

**Returns** *True* if node information is complete, *False* otherwise.

Return type Boolean

See also:

```
AbstractXBeeDevice.read_device_info()
```

# log

Returns the XBee logger.

**Returns** The XBee device logger.

## Return type Logger

### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

## read\_device\_info(init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if True.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

Return type IOSample

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*IOSample* 

## scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

**Return type** Integer

# set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

**Parameters value** (XBee16BitAddress) – New 16-bit address of the XBee.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

# set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

*APIOutputMode* 

### set api output mode value (api output mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• OperationNotSupportedException — If it is not supported by the current protocol.

### See also:

*APIOutputModeBit* 

# set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

### See also:

```
XBee64BitAddress
get_dest_address()
```

## set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*IOLine* 

# set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

# **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*IOLine IOValue* 

# set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

# set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters rate** (Integer) – New IO sampling rate of the XBee in seconds.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

• ATCommandException - If response is not as expected.

### See also:

```
get_io_sampling_rate()
```

## set node id(node id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

### Raises

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_pan_id()
```

# set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
PowerLevel
get_power_level()
```

# set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

#### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

## See also:

```
IOLine
IOMode.PWM
```

## set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

# update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

## **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

## Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

## write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
class digi.xbee.devices.Raw802Device(port=None,
                                                                               baud rate=None,
                                               data_bits=<sphinx.ext.autodoc.importer._MockObject
                                               object>, stop bits=<sphinx.ext.autodoc.importer. MockObject
                                               object>, parity=<sphinx.ext.autodoc.importer._MockObject
                                               object>, flow_control=<FlowControl.NONE: None>,
                                               _sync_ops_timeout=4, comm_iface=None)
     Bases: digi.xbee.devices.XBeeDevice
     This class represents a local 802.15.4 XBee.
     Class constructor. Instantiates a new Raw802Device with the provided parameters.
          Parameters
                 • port (String) - Serial port identifier.
                                                            Depends on operating system.
                                                                                           e.g.
                   '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
                 • baud_rate (Integer) - Serial port baud rate.
                 • (Integer, default (flow_control) - serial.EIGHTBITS): Port bitsize.
                 • (Integer, default - serial.STOPBITS_ONE): Port stop bits.
                 • (Character, default (parity) - serial .PARITY NONE): Port parity.
                 • (Integer, default - FlowControl.NONE): Port flow control.
                   _sync_ops_timeout (Integer, default: 3): Read timeout (in seconds). comm_iface
                       (XBeeCommunicationInterface): Communication interface.
          Raises All exceptions raised by XBeeDevice.__init__() constructor.
     See also:
     XBeeDevice
     XBeeDevice.__init__()
     open (force_settings=False)
          Override.
          See also:
          XBeeDevice.open()
     get_protocol()
          Override.
          See also:
```

# get\_ai\_status()

XBeeDevice.get\_protocol()

Returns the current association status of this XBee. It indicates occurrences of errors during the modem initialization and connection.

## **Returns**

The XBee association indication status.

Return type AssociationIndicationStatus

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## send\_data\_64 (x64addr, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee with the given 64-bit address.

This method waits for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

### **Parameters**

- **x64addr** (XBee64BitAddress) 64-bit address of the destination XBee.
- data (String or Bytearray) Raw data to send.
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## **Returns** The response.

Return type XBeePacket

## Raises

- ValueError If x64addr or data is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

### See also:

XBee64BitAddress XBeePacket

## send\_data\_async\_64 (x64addr, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee with the given 64-bit address.

This method does not wait for a response.

## **Parameters**

- **x64addr** (XBee64BitAddress) 64-bit address of the destination XBee.
- data (String or Bytearray) Raw data to send.

• transmit\_options (Integer, optional) - Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

### Raises

- ValueError If *x64addr* or *data* is *None*.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

## See also:

XBee64BitAddress XBeePacket

# send\_data\_16 (x16addr, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee with the given 16-bit address.

This method will wait for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

#### **Parameters**

- **x16addr** (XBee16BitAddress) 16-bit address of the destination XBee.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## **Returns** The response.

## Return type XBeePacket

# Raises

- ValueError If x16addr or data is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

## See also:

XBee16BitAddress XBeePacket

### send data async 16 (x16addr, data, transmit options=0)

Non-blocking method. This method sends data to a remote XBee with the given 16-bit address.

This method does not wait for a response.

## **Parameters**

- **x16addr** (XBee16BitAddress) 16-bit address of the destination XBee.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

### Raises

- ValueError If x16addr or data is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBee16BitAddress XBeePacket

# add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

## add\_data\_received\_callback(callback)

Adds a callback for the event DataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The data received as an XBeeMessage.

## add\_expl\_data\_received\_callback (callback)

Adds a callback for the event ExplicitDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The explicit data received as a *ExplicitXBeeMessage*.

## add\_fs\_frame\_received\_callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

# See also:

AbstractXBeeDevice FSCmd ReceiveOptions

## add\_io\_sample\_received\_callback (callback)

Adds a callback for the event IOSampleReceived.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The received IO sample as an IOSample.
- The remote XBee which sent the packet as a RemoteXBeeDevice.
- The time in which the packet was received as an Integer.

# add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

## add\_modem\_status\_received\_callback (callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

# add\_packet\_received\_callback (callback)

Adds a callback for the event PacketReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

## add route received callback (callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

Parameters callback (Function) - The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

## See also:

```
XBeeDevice.del_route_received_callback()
```

## add\_socket\_data\_received\_callback(callback)

Adds a callback for the event SocketDataReceived.

Parameters callback (Function) - The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

## add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

• The socket ID as an Integer.

- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

## add\_socket\_state\_received\_callback (callback)

Adds a callback for the event SocketStateReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a *SocketState*.

# add\_user\_data\_relay\_received\_callback(callback)

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

## apply\_changes()

Applies changes via 'AC' command.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)

Applies the given XBee profile to the XBee.

## **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

## **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

## close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

### comm iface

Returns the hardware interface associated to the XBee.

**Returns** Hardware interface of the XBee.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

## classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

"timeout" for -> Timeout for synchronous operations (in seconds).

**Returns** XBee object created.

Return type XBeeDevice

Raises SerialException – If the port to open does not exist or is already opened.

See also:

*XBeeDevice* 

## del\_bluetooth\_data\_received\_callback (callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_data\_received\_callback (callback)

Deletes a callback for the callback list of DataReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_expl\_data\_received\_callback(callback)

Deletes a callback for the callback list of ExplicitDataReceived event.

Parameters callback (Function) - The callback to delete.

# del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_io\_sample\_received\_callback(callback)

Deletes a callback for the callback list of IOSampleReceived event.

<sup>&</sup>quot;port" -> Port number.

<sup>&</sup>quot;bitSize" -> Bit size.

<sup>&</sup>quot;stopBits" -> Stop bits.

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

**Parameters** callback (Function) – The callback to delete.

## del\_micropython\_data\_received\_callback (callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_modem\_status\_received\_callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_packet\_received\_callback (callback)

Deletes a callback for the callback list of PacketReceived event.

Parameters callback (Function) - The callback to delete.

### del route received callback (callback)

Deletes a callback for the callback list of RouteReceived event.

**Parameters** callback (Function) – The callback to delete.

See also:

XBeeDevice.add\_route\_received\_callback()

# del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

## del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of SocketStateReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_user\_data\_relay\_received\_callback(callback)

Deletes a callback for the callback list of RelayDataReceived event.

**Parameters** callback (Function) – The callback to delete.

## determine protocol (hardware version, firmware version)

Determines the XBee protocol based on the given hardware and firmware versions.

# **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (Bytearray) Firmware version to get its protocol.

## **Returns**

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

## disable bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

# enable\_bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method <code>AbstractXBeeDevice.update\_bluetooth\_password()</code>.

Note that your XBee must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# execute\_command (parameter, value=None, apply=None)

Executes the provided command.

## **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is apply changes enabled() returned value.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

AbstractXBeeDevice.get\_parameter()

```
AbstractXBeeDevice.set parameter()
     AbstractXBeeDevice.apply_changes()
     AbstractXBeeDevice.write_changes()
     AbstractXBeeDevice.is_apply_changes_enabled()
     AbstractXBeeDevice.enable_apply_changes()
flush_queues()
    Flushes the packets queue.
get_16bit_addr()
     Returns the 16-bit address of the XBee.
         Returns 16-bit address of the XBee.
         Return type XBee16BitAddress
     See also:
     XBee16BitAddress
get_64bit_addr()
     Returns the 64-bit address of the XBee.
         Returns 64-bit address of the XBee.
         Return type XBee64BitAddress
     See also:
     XBee64BitAddress
get adc value (io line)
     Returns the analog value of the provided IO line.
     The provided IO line must be previously configured as ADC. To do so, use AbstractXBeeDevice.
     set_io_configuration() and IOMode.ADC.
         Parameters io_line (IOLine) - IO line to get its ADC value.
         Returns Analog value corresponding to the provided IO line.
         Return type Integer
         Raises
```

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

## See also:

```
IOLine
set_io_configuration()
```

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

Returns API output mode of the XBee.

Return type APIOutputMode

**Raises** 

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

APIOutputMode

# get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

### get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

**Return type** String

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

Return type Integer

## get dest address()

Returns the 64-bit address of the XBee that is data destination.

**Returns** 64-bit address of destination XBee.

Return type XBee64BitAddress

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
XBee64BitAddress
set_dest_address()
```

# get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

Parameters io\_line (IOLine) - the DIO line to gets its digital value.

Returns current value of the provided IO line.

Return type IOValue

### Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
IOValue
set_io_configuration()
```

## get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

## get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

Return type Bytearray

# get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

HardwareVersion

## get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

Returns IO mode of the IO line provided.

Return type IOMode

**Raises** 

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
set_io_configuration()
```

# get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set_io_sampling_rate()
get_network()
     Returns the network of this XBee.
          Returns The XBee network.
          Return type XBeeNetwork
get_next_frame_id()
     Returns the next frame ID of the XBee.
          Returns The next frame ID of the XBee.
          Return type Integer
get_node_id()
     Returns the node identifier ('NI') value of the XBee.
          Returns Node identifier ('NI') of the XBee.
          Return type String
get pan id()
     Returns the operating PAN ID of the XBee.
          Returns Operating PAN ID of the XBee.
```

Return type Bytearray

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set pan id()
```

get\_parameter (parameter, parameter\_value=None, apply=None)

Override.

See also:

```
AbstractXBeeDevice.get_parameter()
```

```
get_power_level()
```

Returns the power level of the XBee.

**Returns** Power level of the XBee.

Return type PowerLevel

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
set_power_level()
```

# get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

### Raises

• TimeoutException - If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io line has no PWM capability.

### See also:

*IOLine* 

### get\_role()

Gets the XBee role.

Returns the role of the XBee.

Return type Role

See also:

Role

# get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

# For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

# **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

## **Returns**

# **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (None if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

```
Return type Tuple
```

```
get_sync_ops_timeout()
```

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

### get xbee device callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

# has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_packets()
```

## has\_packets()

Returns if there are pending packets to read. This does not include explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_explicit_packets()
```

# $\verb|is_apply_changes_enabled|()$

Returns whether apply changes flag is enabled.

**Returns** *True* if apply changes flag is enabled, *False* otherwise.

Return type Boolean

# is\_device\_info\_complete()

Returns whether XBee node information is complete.

**Returns** *True* if node information is complete, *False* otherwise.

Return type Boolean

See also:

```
AbstractXBeeDevice.read_device_info()
```

## is\_open()

Returns whether this XBee is open.

**Returns** Boolean. *True* if this XBee is open, *False* otherwise.

## is remote()

Override method.

See also:

```
AbstractXBeeDevice.is_remote()
```

# log

Returns the XBee logger.

**Returns** The XBee device logger.

Return type Logger

## operating\_mode

Returns the operating mode of this XBee.

**Returns** OperatingMode. This XBee operating mode.

## reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

# read\_data(timeout=None)

Reads new data received by this XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

**Parameters timeout** (*Integer*, *optional*) – Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

## Returns

Read message or None if this XBee did not receive new data.

Return type XBeeMessage

## **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

# See also:

XBeeMessage

## read\_data\_from (remote\_xbee, timeout=None)

Reads new data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

## **Parameters**

- $remote\_xbee(RemoteXBeeDevice) Remote XBee that sent the data.$
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

## **Returns**

**Read message sent by** *remote\_xbee* or *None* if this XBee did not receive new data.

## Return type XBeeMessage

## **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

#### See also:

XBeeMessage RemoteXBeeDevice

# read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if True.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

AbstractXBeeDevice.is\_device\_info\_complete()

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

```
Return type IOSample
```

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
IOSample
```

## reset()

Override method.

See also:

```
AbstractXBeeDevice.reset()
```

# scan\_counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

# send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

```
Parameters data (Bytearray) – Data to send.
```

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

## See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

## send\_data (remote\_xbee, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee synchronously.

This method will wait for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

## **Parameters**

- remote xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## **Returns** The response.

## Return type XBeePacket

## **Raises**

- ValueError If remote\_xbee is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

### See also:

RemoteXBeeDevice XBeePacket

# send\_data\_async (remote\_xbee, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee.

This method does not wait for a response.

## **Parameters**

- remote\_xbee (RemoteXBeeDevice) the remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## Raises

- ValueError If remote xbee is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

RemoteXBeeDevice

### send data broadcast (data, transmit options=0)

Sends the provided data to all the XBee nodes of the network (broadcast).

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

The received timeout method is configured using AbstractXBeeDevice. and consulted set sync ops timeout() can be with AbstractXBeeDevice. get\_sync\_ops\_timeout() method.

### **Parameters**

- data (String or Bytearray) Data to send.
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

### send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

## See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

# send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

## **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

### Returns

Response packet if sync is True, None otherwise.

Return type XBeePacket

## Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

### See also:

XBeePacket

# send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

Returns Received response packet.

Return type XBeePacket

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

## See also:

XBeePacket

# send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

## **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

## **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

XBeeLocalInterface

### serial port

Returns the serial port associated to the XBee, if any.

#### Returns

Serial port of the XBee. None if the local XBee does not use serial communication.

Return type XBeeSerialPort

See also:

XBeeSerialPort

#### set 16bit addr(value)

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

# $\verb"set_api_output_mode" (api_output_mode")$

Deprecated since version 1.3: Use set api output mode value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• OperationNotSupportedException — If it is not supported by the current protocol.

#### See also:

*APIOutputMode* 

# set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

*APIOutputModeBit* 

# set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

# Raises

- $\bullet$   $\,$  TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

## See also:

```
XBee64BitAddress
get_dest_address()
```

## set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOLine* 

# set\_dio\_value(io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

#### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*IOLine IOValue* 

# $\verb|set_io_configuration|| (io\_line, io\_mode)|$

Sets the configuration of the provided IO line.

## **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
IOLine
IOMode
get_io_configuration()
```

# set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_io_sampling_rate()
```

### set node id(node id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

# Raises

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

# Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
get_pan_id()

set_parameter(parameter, value, apply=None)
   Override.
   See: AbstractXBeeDevice.set_parameter()

set_power_level(power_level)
```

Parameters power\_level (PowerLevel) - New power level of the XBee.

## Raises

Sets the power level of the XBee.

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
PowerLevel
get_power_level()
```

# set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

# **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or cycle is not between 0 and 100.

```
IOLine
IOMode.PWM
```

# set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

## update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_file} & xbee\_firmware\_file=None, \ bootloader\_firmware\_file=None, \ timeout=None, \ progress\_callback=None) \end{tabular}$ 

Performs a firmware update operation of the XBee.

#### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the update process (seconds).
- progress\_callback (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

### write changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code> to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

Bases: digi.xbee.devices.XBeeDevice

This class represents a local DigiMesh XBee.

Class constructor. Instantiates a new <code>DigiMeshDevice</code> with the provided parameters.

#### **Parameters**

- **port** (*String*) serial port identifier. Depends on operating system. e.g. '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
- baud\_rate (Integer) Serial port baud rate.
- (Integer, default (flow\_control) serial.EIGHTBITS): Port bitsize.
- (Integer, default serial.STOPBITS\_ONE): Port stop bits.
- (Character, default (parity) serial.PARITY\_NONE): Port parity.
- (Integer, default FlowControl.NONE): port flow control.

\_sync\_ops\_timeout (Integer, default: 3): Read timeout (in seconds). comm\_iface (XBeeCommunicationInterface): Communication interface.

Raises All exceptions raised by XBeeDevice. \_\_init\_\_() constructor.

```
XBeeDevice
XBeeDevice.__init__()

open (force_settings=False)
    Override.
    See also:

    XBeeDevice.open()

get_protocol()
    Override.
    See also:

    XBeeDevice.get_protocol()
```

# build\_aggregate\_routes()

Forces all nodes in the network to automatically build routes to this node. The receiving node establishes a route back to this node.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

# send\_data\_64 (x64addr, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee with the given 64-bit address.

This method waits for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

### **Parameters**

- **x64addr** (XBee64BitAddress) 64-bit address of the destination XBee.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

**Returns** The response.

Return type XBeePacket

### Raises

• ValueError – If x64addr or data is None.

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

XBee64BitAddress XBeePacket

# send\_data\_async\_64 (x64addr, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee with the given 64-bit address.

This method does not wait for a response.

### **Parameters**

- x64addr (XBee64BitAddress) 64-bit address of the destination XBee.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## Raises

- ValueError If x64addr or data is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

# See also:

XBee64BitAddress XBeePacket

## read\_expl\_data(timeout=None)

Reads new explicit data received by this XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

**Parameters timeout** (Integer, optional) – Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if there is no explicit data available.

### **Returns**

Read message or *None* if this XBee did not receive new explicit data.

Return type ExplicitXBeeMessage

# Raises

• ValueError – If a timeout is specified and is less than 0.

- TimeoutException If a timeout is specified and no explicit data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

ExplicitXBeeMessage

# read\_expl\_data\_from (remote\_xbee, timeout=None)

Reads new explicit data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee that sent the explicit data.
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if there is no data available.

#### Returns

**Read message sent by** *remote\_xbee* or *None* if this XBee did not receive new data from that node.

Return type ExplicitXBeeMessage

# Raises

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no explicit data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

### See also:

ExplicitXBeeMessage
RemoteXBeeDevice

send\_expl\_data(remote\_xbee, data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id, transmit\_options=0)

Blocking method. Sends the provided explicit data to the given XBee, source and destination end points, cluster and profile ids.

This method blocks until a success or error response arrives or the configured receive timeout expires. The default timeout is XBeeDevice.\_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

### **Parameters**

- remote xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

**Returns** Response packet obtained after sending data.

Return type XBeePacket

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

### See also:

RemoteXBeeDevice XBeePacket

send\_expl\_data\_broadcast (data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id, transmit\_options=0)

Sends the provided explicit data to all the XBee nodes of the network (broadcast) using provided source and destination end points, cluster and profile ids.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires. The received timeout is configured using the <code>AbstractXBeeDevice.set\_sync\_ops\_timeout()</code> method and can be consulted with method <code>AbstractXBeeDevice.get\_sync\_ops\_timeout()</code>.

### **Parameters**

- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)

• transmit\_options (Integer, optional) - Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

#### See also:

```
XBeeDevice._send_expl_data()
```

Non-blocking method. Sends the provided explicit data to the given XBee, source and destination end points, cluster and profile ids.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

# See also:

RemoteXBeeDevice

get\_neighbors (neighbor\_cb=None, finished\_cb=None, timeout=None)

Returns the neighbors of this XBee. If *neighbor\_cb* is not defined, the process blocks during the specified timeout.

### **Parameters**

- neighbor\_cb (Function, optional, default=`None`) Method called when a new neighbor is received. Receives two arguments:
  - The XBee that owns this new neighbor.
  - The new neighbor.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives two arguments:
  - The XBee that is searching for its neighbors.
  - A list with the discovered neighbors.
  - An error message if something went wrong.
- timeout (Float, optional, default=`NeighborFinder. DEFAULT\_TIMEOUT`) The timeout in seconds.

### **Returns**

**List of** *Neighbor* **when** *neighbor\_cb* **is not defined,** *None* otherwise (in this case neighbors are received in the callback).

# Return type List

Raises OperationNotSupportedException - If XBee protocol is not DigiMesh.

### See also:

```
com.digi.models.zdo.Neighbor
```

# add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

# add\_data\_received\_callback (callback)

Adds a callback for the event DataReceived.

Parameters callback (Function) - The callback. Receives one argument.

• The data received as an XBeeMessage.

# add\_expl\_data\_received\_callback (callback)

Adds a callback for the event *ExplicitDataReceived*.

Parameters callback (Function) - The callback. Receives one argument.

• The explicit data received as a <code>ExplicitXBeeMessage</code>.

## add\_fs\_frame\_received\_callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.

• Receive options (Integer): Bitfield indicating receive options.

#### See also:

AbstractXBeeDevice FSCmd ReceiveOptions

# add\_io\_sample\_received\_callback(callback)

Adds a callback for the event IOSampleReceived.

**Parameters callback** (Function) – The callback. Receives three arguments.

- The received IO sample as an IOSample.
- The remote XBee which sent the packet as a RemoteXBeeDevice.
- The time in which the packet was received as an Integer.

# add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

Parameters callback (Function) - The callback. Receives one argument.

• The MicroPython data as a Bytearray.

# add\_modem\_status\_received\_callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

# add\_packet\_received\_callback (callback)

Adds a callback for the event PacketReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

# add\_route\_received\_callback (callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

**Parameters** callback (Function) – The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (  $\mbox{\it RemoteXBeeDevice}$  ): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

### See also:

XBeeDevice.del\_route\_received\_callback()

### add\_socket\_data\_received\_callback (callback)

Adds a callback for the event SocketDataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

# add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

# add\_socket\_state\_received\_callback(callback)

Adds a callback for the event SocketStateReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a *SocketState*.

# add\_user\_data\_relay\_received\_callback(callback)

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a UserDataRelayMessage.

# apply\_changes()

Applies changes via 'AC' command.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# apply\_profile (profile\_path, timeout=None, progress\_callback=None)

Applies the given XBee profile to the XBee.

#### **Parameters**

- **profile path** (*String*) Path of the XBee profile file to apply.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

# Raises

• XBeeException – If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

# close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

### comm iface

Returns the hardware interface associated to the XBee.

**Returns** Hardware interface of the XBee.

Return type XBeeCommunicationInterface

### See also:

XBeeCommunicationInterface

# classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

#### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

"timeout" for -> Timeout for synchronous operations (in seconds).

Returns XBee object created.

Return type XBeeDevice

**Raises** SerialException – If the port to open does not exist or is already opened.

## See also:

XBeeDevice

# del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) – The callback to delete.

## del\_data\_received\_callback(callback)

Deletes a callback for the callback list of DataReceived event.

Parameters callback (Function) - The callback to delete.

<sup>&</sup>quot;port" -> Port number.

<sup>&</sup>quot;bitSize" -> Bit size.

<sup>&</sup>quot;stopBits" -> Stop bits.

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

# del\_expl\_data\_received\_callback (callback)

Deletes a callback for the callback list of *ExplicitDataReceived* event.

**Parameters** callback (Function) – The callback to delete.

# del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_io\_sample\_received\_callback(callback)

Deletes a callback for the callback list of IOSampleReceived event.

Parameters callback (Function) - The callback to delete.

## del\_micropython\_data\_received\_callback (callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

Parameters callback (Function) - The callback to delete.

# del\_modem\_status\_received\_callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_packet\_received\_callback(callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

### del route received callback (callback)

Deletes a callback for the callback list of RouteReceived event.

Parameters callback (Function) - The callback to delete.

See also:

XBeeDevice.add\_route\_received\_callback()

## del\_socket\_data\_received\_callback (callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del socket data received from callback (callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

### del\_socket\_state\_received\_callback (callback)

Deletes a callback for the callback list of SocketStateReceived event.

Parameters callback (Function) - The callback to delete.

## del\_user\_data\_relay\_received\_callback(callback)

Deletes a callback for the callback list of RelayDataReceived event.

Parameters callback (Function) - The callback to delete.

# determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

**Parameters** 

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

### disable bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

### enable\_bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method  $AbstractXBeeDevice.update\_bluetooth\_password()$ .

Note that your XBee must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# execute\_command(parameter, value=None, apply=None)

Executes the provided command.

### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- value (bytearray, optional, default=`None`) Command value (if any).
- **apply** (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
AbstractXBeeDevice.get_parameter()
     AbstractXBeeDevice.set parameter()
     AbstractXBeeDevice.apply_changes()
     AbstractXBeeDevice.write changes()
     AbstractXBeeDevice.is_apply_changes_enabled()
     AbstractXBeeDevice.enable_apply_changes()
flush_queues()
    Flushes the packets queue.
get_16bit_addr()
     Returns the 16-bit address of the XBee.
         Returns 16-bit address of the XBee.
         Return type XBee16BitAddress
     See also:
     XBee16BitAddress
get_64bit_addr()
     Returns the 64-bit address of the XBee.
         Returns 64-bit address of the XBee.
         Return type XBee64BitAddress
     See also:
     XBee64BitAddress
get_adc_value(io_line)
     Returns the analog value of the provided IO line.
    The provided IO line must be previously configured as ADC. To do so, use AbstractXBeeDevice.
     set_io_configuration() and IOMode.ADC.
         Parameters io_line (IOLine) - IO line to get its ADC value.
         Returns Analog value corresponding to the provided IO line.
         Return type Integer
         Raises
```

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

```
IOLine
set_io_configuration()
```

# get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

Returns API output mode of the XBee.

Return type APIOutputMode

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*APIOutputMode* 

### get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

**Return type** Bytearray

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

digi.xbee.models.mode.APIOutputModeBit

# get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

Returns The Bluetooth MAC address.

Return type String

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

Return type Integer

# get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

Returns 64-bit address of destination XBee.

Return type XBee64BitAddress

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
XBee64BitAddress
set_dest_address()
```

### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

## See also:

```
IOLine
IOValue
set_io_configuration()
```

# get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

## get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

**Return type** Bytearray

## get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

HardwareVersion

```
get_io_configuration(io_line)
```

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

Returns IO mode of the IO line provided.

Return type IOMode

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
set_io_configuration()
```

# get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_network()
    Returns the network of this XBee.
    Returns The XBee network.
    Return type XBeeNetwork

get_next_frame_id()
    Returns the next frame ID of the XBee.

Returns The next frame ID of the XBee.
```

Return type Integer

```
get_node_id()
     Returns the node identifier ('NI') value of the XBee.
          Returns Node identifier ('NI') of the XBee.
          Return type String
get pan id()
     Returns the operating PAN ID of the XBee.
          Returns Operating PAN ID of the XBee.
          Return type Bytearray
          Raises
                \bullet TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
     See also:
     set pan id()
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
get_power_level()
     Returns the power level of the XBee.
          Returns Power level of the XBee.
          Return type PowerLevel
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
```

• ATCommandException - If response is not as expected.

See also:

```
PowerLevel
set_power_level()
```

# get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

**Return type** Integer

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

### See also:

*IOLine* 

### get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

# get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

# For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

# **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

# **Returns**

# Tuple containing route data:

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

# Return type Tuple

## get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

# get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

# has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_packets()
```

# has\_packets()

Returns if there are pending packets to read. This does not include explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_explicit_packets()
```

# $\verb|is_apply_changes_enabled|()$

Returns whether apply changes flag is enabled.

**Returns** *True* if apply changes flag is enabled, *False* otherwise.

Return type Boolean

# is\_device\_info\_complete()

Returns whether XBee node information is complete.

```
Returns True if node information is complete, False otherwise.
```

Return type Boolean

See also:

```
AbstractXBeeDevice.read device info()
```

# is\_open()

Returns whether this XBee is open.

**Returns** Boolean. *True* if this XBee is open, *False* otherwise.

### is\_remote()

Override method.

See also:

```
AbstractXBeeDevice.is_remote()
```

# log

Returns the XBee logger.

**Returns** The XBee device logger.

Return type Logger

# operating\_mode

Returns the operating mode of this XBee.

**Returns** OperatingMode. This XBee operating mode.

### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

# read\_data(timeout=None)

Reads new data received by this XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

**Parameters timeout** (*Integer*, *optional*) – Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

## Returns

Read message or *None* if this XBee did not receive new data.

Return type XBeeMessage

# Raises

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

XBeeMessage

# read\_data\_from(remote\_xbee, timeout=None)

Reads new data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee that sent the data.
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

### Returns

**Read message sent by** *remote\_xbee* or *None* if this XBee did not receive new data.

Return type XBeeMessage

#### Raises

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

# See also:

XBeeMessage RemoteXBeeDevice

# $\verb"read_device_info" (init=True, fire\_event=True)$

Updates all instance parameters reading them from the XBee.

#### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if *True*.
- **fire\_event** (Boolean, optional, default=`True`) *True* to throw and update event if any parameter changed, *False* otherwise.

## Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
AbstractXBeeDevice.is_device_info_complete()
```

### read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

Returns IO sample read from the XBee.

Return type IOSample

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

# See also:

```
IOSample
```

## reset()

Override method.

See also:

```
AbstractXBeeDevice.reset()
```

### scan\_counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

# send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (Bytearray) – Data to send.

# Raises

• InvalidOperatingModeException – If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

• XBeeException – If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

# send\_data (remote\_xbee, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee synchronously.

This method will wait for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

### **Parameters**

- remote xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

# **Returns** The response.

## Return type XBeePacket

### **Raises**

- ValueError If remote\_xbee is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

#### See also:

```
RemoteXBeeDevice
XBeePacket
```

# send\_data\_async (remote\_xbee, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee.

This method does not wait for a response.

# **Parameters**

- remote\_xbee (RemoteXBeeDevice) the remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## Raises

• ValueError – If remote\_xbee is None.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

RemoteXBeeDevice

# send\_data\_broadcast (data, transmit\_options=0)

Sends the provided data to all the XBee nodes of the network (broadcast).

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

The received timeout is configured using method AbstractXBeeDevice. set\_sync\_ops\_timeout() and can be consulted with AbstractXBeeDevice. get sync ops timeout() method.

#### **Parameters**

- data (String or Bytearray) Data to send.
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

# send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (Bytearray) – Data to send.

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

# See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

# send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

#### **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

### **Returns**

Response packet if sync is True, None otherwise.

Return type XBeePacket

### **Raises**

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

#### See also:

XBeePacket

# send\_packet\_sync\_and\_get\_response (packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

# **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

**Returns** Received response packet.

Return type XBeePacket

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

## See also:

XBeePacket

### send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

#### **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

### See also:

XBeeLocalInterface

# serial\_port

Returns the serial port associated to the XBee, if any.

#### Returns

Serial port of the XBee. *None* if the local XBee does not use serial communication.

Return type XBeeSerialPort

See also:

XBeeSerialPort

# set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

**Parameters value** (XBee16BitAddress) – New 16-bit address of the XBee.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

# set\_api\_output\_mode (api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputMode* 

# set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

*APIOutputModeBit* 

### set dest address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• ValueError - If addr is None.

#### See also:

```
XBee64BitAddress
get_dest_address()
```

# set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOLine* 

# set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

## **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

*IOLine IOValue* 

# set\_io\_configuration (io\_line, io\_mode)

Sets the configuration of the provided IO line.

## **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

### set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters rate** (Integer) – New IO sampling rate of the XBee in seconds.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_io_sampling_rate()
```

## set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) – New node identifier ('NI') of the XBee.

# Raises

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
get_pan_id()
```

## set\_parameter (parameter, value, apply=None)

Override.

```
See: AbstractXBeeDevice.set_parameter()
```

### set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

## set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

## Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or cycle is not between 0 and 100.

#### See also:

IOLine
IOMode.PWM

## set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

## update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### update device data from(device)

Updates the current node information with provided data. This is only for internal use.

Parameters device (AbstractXBeeDevice) - XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_file} & xbee\_firmware\_file=None, \ bootloader\_firmware\_file=None, \ timeout=None, \ progress\_callback=None) \end{tabular}$ 

Performs a firmware update operation of the XBee.

#### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- xbee\_firmware\_file (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.

- timeout (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- progress\_callback (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

#### write changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code> to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

Bases: digi.xbee.devices.XBeeDevice

This class represents a local DigiPoint XBee.

Class constructor. Instantiates a new DigiPointDevice with the provided parameters.

#### **Parameters**

- **port** (*String*) Serial port identifier. Depends on operating system. e.g. '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
- baud\_rate (Integer) Serial port baud rate.

- (Integer, default (\_sync\_ops\_timeout) serial.EIGHTBITS): Port bitsize.
- (Integer, default serial.STOPBITS\_ONE): Port stop bits.
- (Character, default (parity) serial.PARITY\_NONE): Port parity.
- (Integer, default FlowControl.NONE): Port flow control.
- (Integer, default 3): Read timeout (in seconds).
- comm\_iface (XBeeCommunicationInterface) Communication interface.

Raises All exceptions raised by XBeeDevice.\_\_init\_\_() constructor.

### See also:

```
XBeeDevice
XBeeDevice.__init__()

open (force_settings=False)
    Override.
    See also:

    XBeeDevice.open()

get_protocol()
    Override.
    See also:

    XBeeDevice.get_protocol()
```

# $\verb|send_data_64_16| (x64 addr, x16 addr, data, transmit\_options=0)|$

Blocking method. This method sends data to the remote XBee with the given 64-bit/16-bit address.

This method waits for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

## **Parameters**

- **x64addr** (XBee64BitAddress) 64-bit address of the destination XBee.
- **x16addr** (XBee16BitAddress) 16-bit address of the destination XBee, XBee16BitAddress.UNKNOWN ADDRESS if unknown.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

**Returns** The response.

Return type XBeePacket

Raises

- ValueError If x64addr, x16addr or data is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

## See also:

XBee64BitAddress XBee16BitAddress XBeePacket

## send\_data\_async\_64\_16 (x64addr, x16addr, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee with the given 64-bit/16-bit address.

This method does not wait for a response.

#### **Parameters**

- **x64addr** (XBee64BitAddress) 64-bit address of the destination XBee.
- x16addr (XBee16BitAddress) 16-bit address of the destination XBee, XBee16BitAddress.UNKNOWN\_ADDRESS if unknown.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## Raises

- ValueError If x64addr, x16addr or data is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBee64BitAddress XBee16BitAddress XBeePacket

# read\_expl\_data(timeout=None)

Reads new explicit data received by this XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

**Parameters timeout** (Integer, optional) – Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if there is no explicit data available.

### Returns

**Read message or** *None* **if this XBee** did not receive new explicit data.

Return type ExplicitXBeeMessage

### **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no explicit data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

#### See also:

ExplicitXBeeMessage

### read\_expl\_data\_from (remote\_xbee, timeout=None)

Reads new explicit data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee that sent the explicit data.
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if there is no data available.

### Returns

**Read message sent by** *remote\_xbee* or *None* if this XBee did not receive new data from that node.

Return type ExplicitXBeeMessage

## Raises

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no explicit data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

576

ExplicitXBeeMessage
RemoteXBeeDevice

send\_expl\_data(remote\_xbee, data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id, transmit options=0)

Blocking method. Sends the provided explicit data to the given XBee, source and destination end points, cluster and profile ids.

This method blocks until a success or error response arrives or the configured receive timeout expires. The default timeout is XBeeDevice. DEFAULT TIMEOUT SYNC OPERATIONS.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (Integer) Profile ID of the transmission (between 0x0 and 0xFFFF)
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

**Returns** Response packet obtained after sending data.

Return type XBeePacket

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

### See also:

RemoteXBeeDevice XBeePacket

send\_expl\_data\_broadcast (data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id, transmit\_options=0)

Sends the provided explicit data to all the XBee nodes of the network (broadcast) using provided source and destination end points, cluster and profile ids.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires. The received timeout is configured using the <code>AbstractXBeeDevice.set\_sync\_ops\_timeout()</code> method and can be consulted with method <code>AbstractXBeeDevice.get\_sync\_ops\_timeout()</code>.

## **Parameters**

• data (String or Bytearray) - Raw data to send.

- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.
- ValueError if cluster\_id or profile\_id is less than 0x0 or greater than 0xFFFF.

### See also:

```
XBeeDevice._send_expl_data()
```

Non-blocking method. Sends the provided explicit data to the given XBee, source and destination end points, cluster and profile ids.

### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

## See also:

RemoteXBeeDevice

## add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

## add\_data\_received\_callback (callback)

Adds a callback for the event DataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The data received as an XBeeMessage.

## add\_expl\_data\_received\_callback(callback)

Adds a callback for the event ExplicitDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The explicit data received as a *ExplicitXBeeMessage*.

### add fs frame received callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

#### See also:

AbstractXBeeDevice FSCmd ReceiveOptions

## add\_io\_sample\_received\_callback (callback)

Adds a callback for the event IOSampleReceived.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The received IO sample as an IOSample.
- The remote XBee which sent the packet as a RemoteXBeeDevice.
- The time in which the packet was received as an Integer.

## $\verb"add_micropython_data_received_callback" (callback)$

Adds a callback for the event MicroPythonDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

## add\_modem\_status\_received\_callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

## add\_packet\_received\_callback (callback)

Adds a callback for the event PacketReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

## add\_route\_received\_callback (callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

**Parameters** callback (Function) – The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

#### See also:

```
XBeeDevice.del route received callback()
```

## add\_socket\_data\_received\_callback (callback)

Adds a callback for the event SocketDataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

### add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters callback** (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

# $\verb"add_socket_state_received_callback" (callback)$

Adds a callback for the event SocketStateReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a *SocketState*.

### add\_user\_data\_relay\_received\_callback(callback)

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

## apply\_changes()

Applies changes via 'AC' command.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)

Applies the given XBee profile to the XBee.

#### **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) Maximum time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

## close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

### comm iface

Returns the hardware interface associated to the XBee.

Returns Hardware interface of the XBee.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

### classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

## **Parameters**

- **comm\_port\_data** (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

"bitSize" -> Bit size.

"stopBits" -> Stop bits.

"parity" -> Parity.

"flowControl" -> Flow control.

"timeout" for -> Timeout for synchronous operations (in seconds).

### Returns XBee object created.

Return type XBeeDevice

**Raises** SerialException – If the port to open does not exist or is already opened.

#### See also:

#### *XBeeDevice*

## del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) - The callback to delete.

### del\_data\_received\_callback (callback)

Deletes a callback for the callback list of DataReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_expl\_data\_received\_callback(callback)

Deletes a callback for the callback list of ExplicitDataReceived event.

**Parameters** callback (Function) – The callback to delete.

#### del\_fs\_frame\_received\_callback (callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

# ${\tt del\_io\_sample\_received\_callback}\ (callback)$

Deletes a callback for the callback list of IOSampleReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_modem\_status\_received\_callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

Parameters callback (Function) – The callback to delete.

## del\_packet\_received\_callback (callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_route\_received\_callback (callback)

Deletes a callback for the callback list of RouteReceived event.

<sup>&</sup>quot;port" -> Port number.

**Parameters** callback (Function) – The callback to delete.

#### See also:

```
XBeeDevice.add route received callback()
```

### del socket data received callback (callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

# del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of <code>SocketStateReceived</code> event.

**Parameters** callback (Function) – The callback to delete.

## del\_user\_data\_relay\_received\_callback(callback)

Deletes a callback for the callback list of RelayDataReceived event.

Parameters callback (Function) - The callback to delete.

## determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

#### **Returns**

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

## disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

#### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method <code>AbstractXBeeDevice.update\_bluetooth\_password()</code>.

Note that your XBee must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
execute_command (parameter, value=None, apply=None)
```

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()
```

### flush\_queues()

Flushes the packets queue.

```
get_16bit_addr()
```

Returns the 16-bit address of the XBee.

**Returns** 16-bit address of the XBee.

```
Return type XBee16BitAddress
```

See also:

XBee16BitAddress

#### get 64bit addr()

Returns the 64-bit address of the XBee.

**Returns** 64-bit address of the XBee.

Return type XBee64BitAddress

See also:

XBee64BitAddress

### get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set io configuration()</code> and <code>IOMode.ADC</code>.

Parameters io\_line (IOLine) - IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

See also:

```
IOLine
set_io_configuration()
```

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

### Return type APIOutputMode

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*APIOutputMode* 

## get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

digi.xbee.models.mode.APIOutputModeBit

### get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

Returns The Bluetooth MAC address.

Return type String

### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

Return type Integer

### get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

**Returns** 64-bit address of destination XBee.

Return type XBee64BitAddress

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
XBee64BitAddress
set_dest_address()
```

## get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

## Raises

- $\bullet \ \, \hbox{\tt TimeoutException-If response is not received before the read timeout expires.}$
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
     IOValue
     set_io_configuration()
get_file_manager()
     Returns the file system manager for the XBee.
          Returns The file system manager.
          Return type FileSystemManager
          Raises FileSystemNotSupportedException - If the XBee does not support filesys-
get_firmware_version()
     Returns the firmware version of the XBee.
          Returns Firmware version of the XBee.
          Return type Bytearray
get hardware version()
     Returns the hardware version of the XBee.
          Returns Hardware version of the XBee.
          Return type Hardware Version
     See also:
     HardwareVersion
get_io_configuration(io_line)
     Returns the configuration of the provided IO line.
          Parameters io_line (IOLine) – IO line to get its configuration.
          Returns IO mode of the IO line provided.
          Return type IOMode
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                 or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
     See also:
```

*IOLine IOMode* 

set\_io\_configuration()

## get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

#### See also:

```
set_io_sampling_rate()
```

## get\_network()

Returns the network of this XBee.

**Returns** The XBee network.

Return type XBeeNetwork

## get\_next\_frame\_id()

Returns the next frame ID of the XBee.

**Returns** The next frame ID of the XBee.

Return type Integer

## get\_node\_id()

Returns the node identifier ('NI') value of the XBee.

**Returns** Node identifier ('NI') of the XBee.

Return type String

## get\_pan\_id()

Returns the operating PAN ID of the XBee.

**Returns** Operating PAN ID of the XBee.

Return type Bytearray

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set_pan_id()

get_parameter (parameter, parameter_value=None, apply=None)
   Override.
   See also:

AbstractXBeeDevice.get_parameter()
```

### get\_power\_level()

Returns the power level of the XBee.

**Returns** Power level of the XBee.

Return type PowerLevel

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
PowerLevel
set_power_level()
```

## get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) - IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

### See also:

*IOLine* 

```
get_role()
```

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

### get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

## For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

#### **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

## Returns

## **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

Return type Tuple

# get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

### get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

```
has_explicit_packets()
     Returns if there are pending explicit packets to read. This does not include non-explicit packets.
          Returns True if there are pending packets, False otherwise.
          Return type Boolean
     See also:
     XBeeDevice.has_packets()
has_packets()
     Returns if there are pending packets to read. This does not include explicit packets.
          Returns True if there are pending packets, False otherwise.
          Return type Boolean
     See also:
     XBeeDevice.has_explicit_packets()
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is_device_info_complete()
     Returns whether XBee node information is complete.
          Returns True if node information is complete, False otherwise.
          Return type Boolean
     See also:
     AbstractXBeeDevice.read_device_info()
is_open()
     Returns whether this XBee is open.
          Returns Boolean. True if this XBee is open, False otherwise.
is_remote()
     Override method.
     See also:
     AbstractXBeeDevice.is_remote()
```

## log

Returns the XBee logger.

**Returns** The XBee device logger.

Return type Logger

### operating\_mode

Returns the operating mode of this XBee.

**Returns** OperatingMode. This XBee operating mode.

#### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

## read\_data(timeout=None)

Reads new data received by this XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

**Parameters timeout** (*Integer*, *optional*) – Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

#### Returns

Read message or None if this XBee did not receive new data.

Return type XBeeMessage

## Raises

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBeeMessage

# read\_data\_from (remote\_xbee, timeout=None)

Reads new data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee that sent the data.
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

#### Returns

**Read message sent by** *remote\_xbee* or *None* if this XBee did not receive new data.

Return type XBeeMessage

### **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

#### See also:

XBeeMessage RemoteXBeeDevice

### read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if *True*.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

AbstractXBeeDevice.is\_device\_info\_complete()

# ${\tt read\_io\_sample}\,(\,)$

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

Return type IOSample

#### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
IOSample
```

### reset()

Override method.

#### See also:

```
AbstractXBeeDevice.reset()
```

#### scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

**Return type** Integer

## send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (Bytearray) – Data to send.

## Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

## See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

## send\_data (remote\_xbee, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee synchronously.

This method will wait for the packet response. The default timeout is XBeeDevice.

DEFAULT TIMEOUT SYNC OPERATIONS.

### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

**Returns** The response.

### Return type XBeePacket

#### Raises

- ValueError If remote\_xbee is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

### See also:

RemoteXBeeDevice XBeePacket

## send\_data\_async (remote\_xbee, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee.

This method does not wait for a response.

### **Parameters**

- remote\_xbee (RemoteXBeeDevice) the remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

## Raises

- ValueError If remote\_xbee is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

RemoteXBeeDevice

## send\_data\_broadcast (data, transmit\_options=0)

Sends the provided data to all the XBee nodes of the network (broadcast).

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

The received timeout is configured using method AbstractXBeeDevice.  $set\_sync\_ops\_timeout()$  and can be consulted with AbstractXBeeDevice.  $get\_sync\_ops\_timeout()$  method.

#### **Parameters**

• data (String or Bytearray) - Data to send.

• transmit\_options (Integer, optional) - Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

### send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

### See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

### send packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

#### **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

### **Returns**

**Response packet if sync is True, None** otherwise.

Return type XBeePacket

## Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

 XBeeException – If the packet listener is not running or the XBee's communication interface is closed.

#### See also:

XBeePacket

## send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

#### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

**Returns** Received response packet.

Return type XBeePacket

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBeePacket

## send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

## **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface
- data (Bytearray) Data to send.

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $ValueError If local\_interface is None.$
- $\bullet$  XBeeException If there is any problem sending the User Data Relay.

#### See also:

XBeeLocalInterface

### serial\_port

Returns the serial port associated to the XBee, if any.

#### Returns

Serial port of the XBee. None if the local XBee does not use serial communication.

Return type XBeeSerialPort

See also:

XBeeSerialPort

### set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

## set\_api\_output\_mode (api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputMode* 

## set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

**APIOutputModeBit** 

## set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

#### See also:

```
XBee64BitAddress
get_dest_address()
```

# $\verb|set_dio_change_detection| (io\_lines\_set)$

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

**Parameters** io\_lines\_set - Set of IOLine.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

• ATCommandException - If response is not as expected.

#### See also:

*IOLine* 

### set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

IOLine IOValue

## set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
IOLine
IOMode
get_io_configuration()
```

### set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
get_io_sampling_rate()
```

### set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

#### Raises

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (*Bytearray*) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_pan_id()
```

```
set_parameter (parameter, value, apply=None)
Override.
```

```
See: AbstractXBeeDevice.set_parameter()
```

## set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

## set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

#### **Parameters**

- io line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or cycle is not between 0 and 100.

### See also:

```
IOLine
IOMode.PWM
```

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

### update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

Performs a firmware update operation of the XBee.

### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- progress\_callback (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

### write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code> to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

Bases: digi.xbee.devices.XBeeDevice

This class represents a local Zigbee XBee.

Class constructor. Instantiates a new ZigBeeDevice with the provided parameters.

### **Parameters**

- **port** (*String*) Serial port identifier. Depends on operating system. e.g '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
- baud\_rate (Integer) Serial port baud rate.
- (Integer, default (flow\_control) serial.EIGHTBITS): Port bitsize.
- (Integer, default serial.STOPBITS\_ONE): Port stop bits.
- (Character, default (parity) serial.PARITY\_NONE): Port parity.
- (Integer, default FlowControl.NONE): Port flow control.

\_sync\_ops\_timeout (Integer, default: 3): Read timeout (in seconds). comm\_iface (XBeeCommunicationInterface): Communication interface.

**Raises** All exceptions raised by XBeeDevice.\_\_init\_\_() constructor.

## See also:

```
XBeeDevice
XBeeDevice.__init__()

open (force_settings=False)
    Override.
See also:
```

```
XBeeDevice.open()
```

## get\_protocol()

Override.

See also:

```
XBeeDevice.get_protocol()
```

### get\_ai\_status()

Returns the current association status of this XBee. It indicates occurrences of errors during the modem initialization and connection.

#### Returns

The XBee association indication status.

Return type AssociationIndicationStatus

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## force\_disassociate()

Forces this XBee to immediately disassociate from the network and re-attempt to associate.

Only valid for End Devices.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_many\_to\_one\_broadcasting\_time()

Returns the time between aggregation route broadcast in tenths of a second.

### Returns

**The number of tenths of a second between aggregation route** broadcasts. -1 if it is disabled.

### Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# set\_many\_to\_one\_broadcasting\_time (tenths\_second)

Configures the time between aggregation route broadcast in tenths of a second.

**Parameters tenths\_second** (*Integer*) – The number of tenths of a second between aggregation route broadcasts. -1 to disable. 0 to only send one broadcast.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If tenths second is None or is lower than -1, or bigger than 254.

### send\_data\_64\_16 (x64addr, x16addr, data, transmit\_options=0)

Blocking method. This method sends data to the remote XBee with the given 64-bit/16-bit address.

This method waits for the packet response. The default timeout is XBeeDevice. DEFAULT TIMEOUT SYNC OPERATIONS.

#### **Parameters**

- **x64addr** (XBee64BitAddress) 64-bit address of the destination XBee.
- x16addr (XBee16BitAddress) 16-bit address of the destination XBee, XBee16BitAddress.UNKNOWN\_ADDRESS if unknown.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

### **Returns** The response.

Return type XBeePacket

#### Raises

- ValueError If x64addr, x16addr or data is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

## See also:

XBee64BitAddress XBee16BitAddress XBeePacket

## send\_data\_async\_64\_16 (x64addr, x16addr, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee with the given 64-bit/16-bit address.

This method does not wait for a response.

#### **Parameters**

- **x64addr** (XBee64BitAddress) 64-bit address of the destination XBee.
- x16addr (XBee16BitAddress) 16-bit address of the destination XBee, XBee16BitAddress.UNKNOWN ADDRESS if unknown.
- data (String or Bytearray) Raw data to send.
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- ValueError If x64addr, x16addr or data is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBee64BitAddress XBee16BitAddress XBeePacket

## read\_expl\_data(timeout=None)

Reads new explicit data received by this XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

**Parameters timeout** (*Integer*, *optional*) – Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if there is no explicit data available.

# Returns

**Read message or** *None* **if this XBee** did not receive new explicit data.

Return type ExplicitXBeeMessage

## **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no explicit data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

# See also:

ExplicitXBeeMessage

608

#### read expl data from(remote xbee, timeout=None)

Reads new explicit data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee that sent the explicit data.
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if there is no data available.

### **Returns**

**Read message sent by** *remote\_xbee* or *None* if this XBee did not receive new data from that node.

Return type ExplicitXBeeMessage

### **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no explicit data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

#### See also:

ExplicitXBeeMessage
RemoteXBeeDevice

send\_expl\_data(remote\_xbee, data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id, transmit\_options=0)

Blocking method. Sends the provided explicit data to the given XBee, source and destination end points, cluster and profile ids.

This method blocks until a success or error response arrives or the configured receive timeout expires. The default timeout is XBeeDevice.\_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (Integer) Profile ID of the transmission (between 0x0 and 0xFFFF)
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

**Returns** Response packet obtained after sending data.

Return type XBeePacket

## Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

#### See also:

RemoteXBeeDevice XBeePacket

send\_expl\_data\_broadcast (data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id, transmit options=0)

Sends the provided explicit data to all the XBee nodes of the network (broadcast) using provided source and destination end points, cluster and profile ids.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires. The received timeout is configured using the  $AbstractXBeeDevice.set\_sync\_ops\_timeout()$  method and can be consulted with method  $AbstractXBeeDevice.get\_sync\_ops\_timeout()$ .

#### **Parameters**

- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

### See also:

XBeeDevice.\_send\_expl\_data()

Non-blocking method. Sends the provided explicit data to the given XBee, source and destination end points, cluster and profile ids.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)
- **transmit\_options** (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.
- ValueError if *cluster\_id* or *profile\_id* is less than 0x0 or greater than 0xFFFF.

# See also:

RemoteXBeeDevice

**send\_multicast\_data** (group\_id, data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id)

Blocking method. This method sends multicast data to the provided group ID synchronously.

This method will wait for the packet response. The default timeout for this method is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

#### **Parameters**

- **group** id (XBee16BitAddress) 16-bit address of the multicast group.
- data (Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (Integer) Profile ID of the transmission (between 0x0 and 0xFFFF)

**Returns** the response packet.

## Return type XBeePacket

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

#### See also:

XBee16BitAddress XBeePacket

send\_multicast\_data\_async(group\_id, data, src\_endpoint, dest\_endpoint, cluster\_id, profile id)

Non-blocking method. This method sends multicast data to the provided group ID.

This method does not wait for a response.

## **Parameters**

- group\_id (XBee16BitAddress) 16-bit address of the multicast group.
- data (Bytearray) Raw data to send.
- **src\_endpoint** (*Integer*) Source endpoint of the transmission. 1 byte.
- **dest\_endpoint** (*Integer*) Destination endpoint of the transmission. 1 byte.
- cluster\_id (Integer) Cluster ID of the transmission (between 0x0 and 0xFFFF)
- **profile\_id** (*Integer*) Profile ID of the transmission (between 0x0 and 0xFFFF)

# Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

## See also:

XBee16BitAddress

# register\_joining\_device (registrant\_address, options, key)

Securely registers a joining device to a trust center. Registration is the process by which a node is authorized to join the network using a preconfigured link key or installation code that is conveyed to the trust center out-of-band (using a physical interface and not over-the-air).

This method is synchronous, it sends the register joining device request and waits for the answer of the operation. Then, returns the corresponding status.

#### **Parameters**

- registrant\_address (XBee64BitAddress) 64-bit address of the device to register.
- options (RegisterKeyOptions) Register options indicating the key source.
- **key** (*Bytearray*) Key of the device to register.

### Returns

**Register device operation status or** *None* if the answer is not a *RegisterDeviceStatus-Packet*.

Return type ZigbeeRegisterStatus

### **Raises**

- TimeoutException If the answer is not received in the configured timeout.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.
- ValueError If registrant\_address or options is None.

#### See also:

RegisterKeyOptions XBee64BitAddress ZigbeeRegisterStatus

## register\_joining\_device\_async (registrant\_address, options, key)

Securely registers a joining device to a trust center. Registration is the process by which a node is authorized to join the network using a preconfigured link key or installation code that is conveyed to the trust center out-of-band (using a physical interface and not over-the-air).

This method is asynchronous, which means that it does not wait for an answer after sending the request.

### **Parameters**

- registrant\_address (XBee64BitAddress) 64-bit address of the device to register.
- options (RegisterKeyOptions) Register options indicating the key source.
- **key** (*Bytearray*) Key of the device to register.

## **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.
- ValueError if registrant\_address or options is None.

## See also:

RegisterKeyOptions XBee64BitAddress

## unregister\_joining\_device (unregistrant\_address)

Unregisters a joining device from a trust center.

This method is synchronous, it sends the unregister joining device request and waits for the answer of the operation. Then, returns the corresponding status.

**Parameters unregistrant\_address** (XBee64BitAddress) – 64-bit address of the device to unregister.

#### Returns

**Unregister device operation status** or *None* if the answer is not a *RegisterDeviceStatus-Packet*.

Return type ZigbeeRegisterStatus

#### Raises

- TimeoutException If the answer is not received in the configured timeout.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.
- ValueError If registrant\_address is None.

### See also:

XBee64BitAddress ZigbeeRegisterStatus

# unregister\_joining\_device\_async (unregistrant\_address)

Unregisters a joining device from a trust center.

This method is asynchronous, which means that it will not wait for an answer after sending the unregister request.

**Parameters unregistrant\_address** (XBee64BitAddress) - 64-bit address of the device to unregister.

# Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.
- ValueError If registrant\_address is None.

## See also:

XBee64BitAddress

# $\verb"get_routes" (route\_cb=None, finished\_cb=None, timeout=None)"$

Returns the routes of this XBee. If *route\_cb* is not defined, the process blocks until the complete routing table is read.

# **Parameters**

- route\_cb (Function, optional, default=`None`) Method called when a new route is received. Receives two arguments:
  - The XBee that owns this new route.
  - The new route.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that executed the ZDO command.
  - A list with the discovered routes.
  - An error message if something went wrong.
- timeout (Float, optional, default=`RouteTableReader.

  DEFAULT TIMEOUT`) The ZDO command timeout in seconds.

### **Returns**

**List of** *Route* **when** *route\_cb* **is not defined,** *None* **otherwise (in this case routes are received in the callback).** 

### **Return type** List

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee is not Zigbee or Smart Energy.
- XBeeException If the XBee's communication interface is closed.

## See also:

```
com.digi.models.zdo.Route
```

# get\_neighbors (neighbor\_cb=None, finished\_cb=None, timeout=None)

Returns the neighbors of this XBee. If *neighbor\_cb* is not defined, the process blocks until the complete neighbor table is read.

#### **Parameters**

- neighbor\_cb (Function, optional, default=`None`) Method called when a new neighbor is received. Receives two arguments:
  - The XBee that owns this new neighbor.
  - The new neighbor.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that executed the ZDO command.
  - A list with the discovered neighbors.
  - An error message if something went wrong.
- timeout (Float, optional, default=`NeighborTableReader. DEFAULT TIMEOUT`) The ZDO command timeout in seconds.

#### **Returns**

**List of** *Neighbor* **when** *neighbor\_cb* **is not defined,** *None* otherwise (in this case neighbors are received in the callback).

# Return type List

Raises OperationNotSupportedException - If XBee is not Zigbee or Smart Energy.

#### See also:

com.digi.models.zdo.Neighbor

## create\_source\_route (dest\_node, hops)

Creates a source route for the provided destination node. A source route specifies the complete route a packet traverses to get from source to destination.

For best results, use source routing with many-to-one routing.

#### **Parameters**

- **dest\_node** (*RemoteXBeeDevice*) The destination node.
- hops (List) List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination excluded).

#### Raises

- ValueError If dest\_node is None, or if it is a local node, or if its protocol is not Zigbee based, or if its 64-bit address or 16-bit address is None, unknown, or invalid.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

## add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

# add\_data\_received\_callback (callback)

Adds a callback for the event DataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The data received as an XBeeMessage.

# add\_expl\_data\_received\_callback (callback)

Adds a callback for the event ExplicitDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The explicit data received as a *ExplicitXBeeMessage*.

## add\_fs\_frame\_received\_callback (callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

• Source (AbstractXBeeDevice): The node that sent the file system frame.

- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

## See also:

AbstractXBeeDevice FSCmd ReceiveOptions

### add\_io\_sample\_received\_callback (callback)

Adds a callback for the event IOSampleReceived.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The received IO sample as an IOSample.
- The remote XBee which sent the packet as a RemoteXBeeDevice.
- The time in which the packet was received as an Integer.

# add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

### add\_modem\_status\_received\_callback (callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

## add\_packet\_received\_callback (callback)

Adds a callback for the event PacketReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

# add\_route\_received\_callback(callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

 $\begin{tabular}{ll} \textbf{Parameters} & \textbf{callback} & (\textit{Function}) - \textbf{The callback}. & \textbf{Receives three arguments}. \\ \end{tabular}$ 

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

### See also:

```
XBeeDevice.del_route_received_callback()
```

#### add socket data received callback (callback)

Adds a callback for the event Socket DataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

### add socket data received from callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

## add\_socket\_state\_received\_callback(callback)

Adds a callback for the event SocketStateReceived.

**Parameters callback** (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a *SocketState*.

# add\_user\_data\_relay\_received\_callback(callback)

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

# apply\_changes()

Applies changes via 'AC' command.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### apply profile (profile path, timeout=None, progress callback=None)

Applies the given XBee profile to the XBee.

### **Parameters**

- **profile\_path** (String) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

### **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

#### close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

#### comm iface

Returns the hardware interface associated to the XBee.

**Returns** Hardware interface of the XBee.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

## classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

#### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

**Returns** XBee object created.

Return type XBeeDevice

Raises SerialException – If the port to open does not exist or is already opened.

See also:

XBeeDevice

# del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) - The callback to delete.

<sup>&</sup>quot;port" -> Port number.

<sup>&</sup>quot;bitSize" -> Bit size.

<sup>&</sup>quot;stopBits" -> Stop bits.

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

<sup>&</sup>quot;timeout" for -> Timeout for synchronous operations (in seconds).

#### del data received callback (callback)

Deletes a callback for the callback list of DataReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_expl\_data\_received\_callback (callback)

Deletes a callback for the callback list of *ExplicitDataReceived* event.

**Parameters** callback (Function) – The callback to delete.

# del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_io\_sample\_received\_callback (callback)

Deletes a callback for the callback list of IOSampleReceived event.

Parameters callback (Function) - The callback to delete.

# del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_modem\_status\_received\_callback (callback)

Deletes a callback for the callback list of ModemStatusReceived event.

**Parameters** callback (Function) – The callback to delete.

#### del packet received callback(callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_route\_received\_callback (callback)

Deletes a callback for the callback list of RouteReceived event.

**Parameters** callback (Function) – The callback to delete.

See also:

XBeeDevice.add\_route\_received\_callback()

### del socket data received callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_socket\_data\_received\_from\_callback (callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

## del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of *SocketStateReceived* event.

**Parameters** callback (Function) – The callback to delete.

# del\_user\_data\_relay\_received\_callback (callback)

Deletes a callback for the callback list of RelayDataReceived event.

**Parameters** callback (Function) – The callback to delete.

#### determine protocol (hardware version, firmware version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

#### Returns

XBee protocol corresponding to the given hardware and firmware versions.

Return type XBeeProtocol

## disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

# enable\_bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method <code>AbstractXBeeDevice.update\_bluetooth\_password()</code>.

Note that your XBee must include Bluetooth Low Energy support.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# execute\_command(parameter, value=None, apply=None)

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- value (bytearray, optional, default=`None`) Command value (if any).

• apply (Boolean, optional, default=`None`) - True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.get_parameter()
    AbstractXBeeDevice.set_parameter()
    AbstractXBeeDevice.apply_changes()
    AbstractXBeeDevice.write_changes()
    AbstractXBeeDevice.is_apply_changes_enabled()
    AbstractXBeeDevice.enable apply changes()
flush_queues()
    Flushes the packets queue.
get_16bit_addr()
    Returns the 16-bit address of the XBee.
         Returns 16-bit address of the XBee.
         Return type XBee16BitAddress
    See also:
    XBee16BitAddress
get 64bit addr()
    Returns the 64-bit address of the XBee.
         Returns 64-bit address of the XBee.
         Return type XBee64BitAddress
    See also:
```

## get\_adc\_value(io\_line)

XBee64BitAddress

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use AbstractXBeeDevice.  $set\_io\_configuration()$  and IOMode.ADC.

**Parameters** io\_line (*IOLine*) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
set_io_configuration()
```

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
APIOutputMode
```

### get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

**Return type** Bytearray

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

#### get bluetooth mac addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

**Return type** String

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

**Return type** Integer

### get dest address()

Returns the 64-bit address of the XBee that is data destination.

**Returns** 64-bit address of destination XBee.

Return type XBee64BitAddress

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
XBee64BitAddress
set_dest_address()
```

### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

## See also:

```
IOLine
IOValue
set_io_configuration()
```

# get\_file\_manager()

Returns the file system manager for the XBee.

Returns The file system manager.

Return type FileSystemManager

 $\label{lem:Raises} \textbf{Raises} \ \texttt{FileSystemNotSupportedException} - If \ the \ XBee \ does \ not \ support \ filesystem.$ 

# get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

**Return type** Bytearray

## get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

```
HardwareVersion
```

```
get_io_configuration(io_line)
```

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

Returns IO mode of the IO line provided.

Return type IOMode

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
IOLine
IOMode
set_io_configuration()
```

## get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

Returns IO sampling rate of XBee.

Return type Integer

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
get_network()
    Returns the network of this XBee.
    Returns The XBee network.
    Return type XBeeNetwork
get_next_frame_id()
```

Returns the next frame ID of the XBee.

626

```
Returns The next frame ID of the XBee.
```

**Return type** Integer

```
get_node_id()
```

Returns the node identifier ('NI') value of the XBee.

**Returns** Node identifier ('NI') of the XBee.

Return type String

### get\_pan\_id()

Returns the operating PAN ID of the XBee.

Returns Operating PAN ID of the XBee.

**Return type** Bytearray

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
set_pan_id()
```

get\_parameter (parameter, parameter\_value=None, apply=None)

Override.

See also:

```
AbstractXBeeDevice.get_parameter()
```

# get\_power\_level()

Returns the power level of the XBee.

**Returns** Power level of the XBee.

Return type PowerLevel

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
PowerLevel
set_power_level()
```

#### get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) - IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

### See also:

*IOLine* 

# get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

## get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

# For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

# **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.

• **force** (Boolean) – *True* to force asking for the route, *False* otherwise. Only for Zigbee.

#### Returns

# **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

# Return type Tuple

# get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

## get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

```
Returns PacketReceived
```

# has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_packets()
```

## has packets()

Returns if there are pending packets to read. This does not include explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_explicit_packets()
```

# is\_apply\_changes\_enabled()

Returns whether apply changes flag is enabled.

```
Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is_device_info_complete()
     Returns whether XBee node information is complete.
          Returns True if node information is complete, False otherwise.
          Return type Boolean
     See also:
     AbstractXBeeDevice.read device info()
is_open()
     Returns whether this XBee is open.
          Returns Boolean. True if this XBee is open, False otherwise.
is_remote()
     Override method.
     See also:
     AbstractXBeeDevice.is_remote()
log
     Returns the XBee logger.
          Returns The XBee device logger.
          Return type Logger
operating_mode
     Returns the operating mode of this XBee.
          Returns OperatingMode. This XBee operating mode.
reachable
     Returns whether the XBee is reachable.
          Returns True if the device is reachable, False otherwise.
          Return type Boolean
read data(timeout=None)
     Reads new data received by this XBee.
     If timeout is specified, this method blocks until new data is received or the timeout expires, throwing a
      TimeoutException in this case.
          Parameters timeout (Integer, optional) - Read timeout in seconds. If None, this
              method is non-blocking and returns None if no data is available.
          Returns
              Read message or None if this XBee did not receive new data.
          Return type XBeeMessage
```

## **Raises**

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

### See also:

XBeeMessage

# read\_data\_from (remote\_xbee, timeout=None)

Reads new data received from the given remote XBee.

If *timeout* is specified, this method blocks until new data is received or the timeout expires, throwing a *TimeoutException* in this case.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee that sent the data.
- **timeout** (*Integer*, *optional*) Read timeout in seconds. If *None*, this method is non-blocking and returns *None* if no data is available.

### **Returns**

Read message sent by remote\_xbee or None if this XBee did not receive new data.

# Return type XBeeMessage

#### Raises

- ValueError If a timeout is specified and is less than 0.
- TimeoutException If a timeout is specified and no data was received during that time.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBeeMessage

RemoteXBeeDevice

# read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

#### **Parameters**

• init (Boolean, optional, default=`True`) – If False only not initialized parameters are read, all if True.

• **fire\_event** (Boolean, optional, default=`True`) – *True* to throw and update event if any parameter changed, *False* otherwise.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

# read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

```
Return type IOSample
```

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
IOSample
```

### reset()

Override method.

# See also:

```
AbstractXBeeDevice.reset()
```

#### scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

# send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

## send\_data (remote\_xbee, data, transmit\_options=0)

Blocking method. This method sends data to a remote XBee synchronously.

This method will wait for the packet response. The default timeout is XBeeDevice. \_DEFAULT\_TIMEOUT\_SYNC\_OPERATIONS.

#### **Parameters**

- remote\_xbee (RemoteXBeeDevice) Remote XBee to send data to.
- data (String or Bytearray) Raw data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

**Returns** The response.

Return type XBeePacket

#### Raises

- ValueError If remote\_xbee is None.
- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- XBeeException If the XBee's communication interface is closed.

## See also:

```
RemoteXBeeDevice
XBeePacket
```

# send\_data\_async (remote\_xbee, data, transmit\_options=0)

Non-blocking method. This method sends data to a remote XBee.

This method does not wait for a response.

### **Parameters**

- remote\_xbee (RemoteXBeeDevice) the remote XBee to send data to.
- data (String or Bytearray) Raw data to send.

• transmit\_options (Integer, optional) - Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- ValueError If remote xbee is None.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee's communication interface is closed.

### See also:

RemoteXBeeDevice

# send\_data\_broadcast (data, transmit\_options=0)

Sends the provided data to all the XBee nodes of the network (broadcast).

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

The received AbstractXBeeDevice. timeout is configured using method set\_sync\_ops\_timeout() and can consulted with AbstractXBeeDevice. be get sync ops timeout () method.

#### **Parameters**

- data (String or Bytearray) Data to send.
- transmit\_options (Integer, optional) Transmit options, bitfield of TransmitOptions. Default to TransmitOptions.NONE.value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TransmitException If the status of the response received is not OK.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

### send micropython data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

# Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

### send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

### **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

### Returns

Response packet if sync is True, None otherwise.

Return type XBeePacket

#### Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

# See also:

XBeePacket

## send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

# **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

**Returns** Received response packet.

Return type XBeePacket

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBeePacket

# send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

### **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

#### See also:

XBeeLocalInterface

## serial\_port

Returns the serial port associated to the XBee, if any.

#### Returns

Serial port of the XBee. None if the local XBee does not use serial communication.

Return type XBeeSerialPort

#### See also:

*XBeeSerialPort* 

# set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

### set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputMode* 

#### set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- $\bullet$  OperationNotSupportedException If it is not supported by the current protocol.

### See also:

APIOutputModeBit

## set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

# Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

### See also:

```
XBee64BitAddress
get_dest_address()
```

# set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOLine* 

# set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

## **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*IOLine IOValue* 

### set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

#### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

## set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_io_sampling_rate()
```

# set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

### **Raises**

- ValueError If *node id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
get_pan_id()
```

set\_parameter (parameter, value, apply=None)

Override.

**See:** AbstractXBeeDevice.set\_parameter()

# set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

# set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

#### **Parameters**

• io\_line (IOLine) – IO Line to be assigned.

• cycle (Integer) – Duty cycle in % to be assigned. Must be between 0 and 100.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

#### See also:

```
IOLine
IOMode.PWM
```

## set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

# update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

 $\label{lem:continuous} \begin{tabular}{ll} \textbf{update\_firmware\_file}, & xbee\_firmware\_file=None, & bootloader\_firmware\_file=None, \\ & timeout=None, & progress\_callback=None) \end{tabular}$ 

Performs a firmware update operation of the XBee.

### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.

- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

### **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

# write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

**class** digi.xbee.devices.**IPDevice**(port=None,

baud rate=None,

data\_bits=<sphinx.ext.autodoc.importer.\_MockObject object>, stop\_bits=<sphinx.ext.autodoc.importer.\_MockObject object>, parity=<sphinx.ext.autodoc.importer.\_MockObject object>, flow\_control=<FlowControl.NONE: None>, \_sync\_ops\_timeout=4, comm\_iface=None)

Bases: digi.xbee.devices.XBeeDevice

This class provides common functionality for XBee IP devices.

Class constructor. Instantiates a new IPDevice with the provided parameters.

# Parameters

• **port** (*String*) – Serial port identifier. Depends on operating system. e.g. '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.

```
• baud_rate (Integer) - Serial port baud rate.
           • (Integer, default (_sync_ops_timeout) - serial.EIGHTBITS): Port
             bitsize.
           • (Integer, default - serial.STOPBITS_ONE): Port stop bits.
           • (Character, default (parity) - serial.PARITY NONE): Port parity.
           • (Integer, default - FlowControl.NONE): Port flow control.
           • (Integer, default – 3): Read timeout (in seconds).
           • comm_iface (XBeeCommunicationInterface) - Communication interface.
    Raises All exceptions raised by XBeeDevice.__init__() constructor.
XBeeDevice
XBeeDevice.__init__()
is_device_info_complete()
     Override.
    See also:
    AbstractXBeeDevice.is_device_info_complete()
```

### get\_ip\_addr()

See also:

Returns the IP address of this IP XBee.

To refresh this value use the method IPDevice.read device info().

**Returns** The IP address of this IP device.

Return type ipaddress. IPv4Address

See also:

ipaddress.IPv4Address

# set\_dest\_ip\_addr (address)

Sets the destination IP address.

Parameters address (ipaddress.IPv4Address) – Destination IP address.

### Raises

- ValueError If address is None.
- TimeoutException If there is a timeout setting the destination IP address.
- XBeeException If there is any other XBee related exception.

See also:

```
ipaddress.IPv4Address
```

## get\_dest\_ip\_addr()

Returns the destination IP address.

**Returns** Configured destination IP address.

Return type ipaddress. IPv4Address

### Raises

- TimeoutException If there is a timeout getting the destination IP address.
- XBeeException If there is any other XBee related exception.

#### See also:

ipaddress. IPv4Address

### add\_ip\_data\_received\_callback (callback)

Adds a callback for the event IPDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The data received as an IPMessage

#### del ip data received callback (callback)

Deletes a callback for the callback list of IPDataReceived event.

**Parameters** callback (Function) – The callback to delete.

## start\_listening(src\_port)

Starts listening for incoming IP transmissions in the provided port.

**Parameters** src\_port (Integer) – Port to listen for incoming transmissions.

### **Raises**

- ValueError If *source\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout setting the source port.
- XBeeException If there is any other XBee related exception.

### stop\_listening()

Stops listening for incoming IP transmissions.

### **Raises**

- TimeoutException If there is a timeout processing the operation.
- $\bullet$  XBeeException If there is any other XBee related exception.

### send\_ip\_data(ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port using the specified IP protocol. For TCP and TCP SSL protocols, you can also indicate if the socket should be closed when data is sent.

This method blocks until a success or error response arrives or the configured receive timeout expires.

### **Parameters**

• ip addr (ipaddress.IPv4Address) - The IP address to send IP data to.

- **dest\_port** (*Integer*) The destination port of the transmission.
- **protocol** (*IPProtocol*) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- **close\_socket** (Boolean, optional, default=`False`) True to close the socket just after the transmission. False to keep it open.

#### Raises

- ValueError If ip\_addr or protocol or data is None or dest\_port is less than 0 or greater than 65535.
- OperationNotSupportedException If the XBee is remote.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

### send\_ip\_data\_async (ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port asynchronously using the specified IP protocol. For TCP and TCP SSL protocols, you can also indicate if the socket should be closed when data is sent.

Asynchronous transmissions do not wait for answer from the remote device or for transmit status packet.

#### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- protocol (IPProtocol) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- **close\_socket** (Boolean, optional, default=`False`) True to close the socket just after the transmission. False to keep it open.

#### Raises

- ValueError If *ip\_addr* or *protocol* or *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- OperationNotSupportedException If the XBee is remote.
- XBeeException If there is any other XBee related exception.

# send\_ip\_data\_broadcast(dest\_port, data)

Sends the provided IP data to all clients.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

### **Parameters**

- **dest\_port** (*Integer*) The destination port of the transmission.
- data (String or Bytearray) The IP data to be sent.

#### Raises

- ValueError If *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

```
read ip data(timeout=3)
```

Reads new IP data received by this XBee during the provided timeout.

This method blocks until new IP data is received or the provided timeout expires.

For non-blocking operations, register a callback and use the method *IPDevice*. add\_ip\_data\_received\_callback().

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method <code>IPDevice.start\_listening()</code> for that purpose. When finished, you can use the method <code>IPDevice.stop\_listening()</code> to stop listening for incoming IP data.

Parameters timeout (Integer, optional) - The time to wait for new IP data in seconds.

**Returns** IP message, *None* if this device did not receive new data.

Return type IPMessage

**Raises** ValueError – If timeout is less than 0.

### read\_ip\_data\_from(ip\_addr, timeout=3)

Reads new IP data received from the given IP address during the provided timeout.

This method blocks until new IP data from the provided IP address is received or the given timeout expires.

For non-blocking operations, register a callback and use the method IPDevice add\_ip\_data\_received\_callback().

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop\_listening() to stop listening for incoming IP data.

## **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to read data from.
- **timeout** (Integer, optional) The time to wait for new IP data in seconds.

### **Returns**

**IP** message, *None* if this device did not receive new data from the provided IP address.

Return type IPMessage

**Raises** ValueError – If timeout is less than 0.

# get\_network()

Deprecated.

This protocol does not support the network functionality.

#### get\_16bit\_addr()

Deprecated.

This protocol does not have an associated 16-bit address.

# get\_dest\_address()

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.get\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

## set\_dest\_address(addr)

Deprecated.

Operation not supported in this protocol. Use IPDevice.set\_dest\_ip\_addr() instead. This method raises an AttributeError.

### get\_pan\_id()

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### set\_pan\_id(value)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### add\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### del\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### add\_expl\_data\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### del\_expl\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### read\_data(timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### read\_data\_from (remote\_xbee, timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## send\_data\_broadcast (data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### send\_data (remote\_xbee, data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## send\_data\_async (remote\_xbee, data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

Parameters callback (Function) - The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

## add\_fs\_frame\_received\_callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

#### See also:

AbstractXBeeDevice FSCmd ReceiveOptions

### add\_io\_sample\_received\_callback (callback)

Adds a callback for the event IOSampleReceived.

**Parameters callback** (Function) – The callback. Receives three arguments.

- The received IO sample as an IOSample.
- The remote XBee which sent the packet as a RemoteXBeeDevice.
- The time in which the packet was received as an Integer.

# add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

## add\_modem\_status\_received\_callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

### add\_packet\_received\_callback (callback)

Adds a callback for the event PacketReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

### add\_route\_received\_callback(callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

**Parameters** callback (Function) – The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

### See also:

XBeeDevice.del\_route\_received\_callback()

#### add socket data received callback (callback)

Adds a callback for the event Socket DataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

### add socket data received from callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

### add\_socket\_state\_received\_callback(callback)

Adds a callback for the event SocketStateReceived.

**Parameters callback** (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a SocketState.

## add\_user\_data\_relay\_received\_callback(callback)

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

## apply\_changes()

Applies changes via 'AC' command.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### apply profile (profile path, timeout=None, progress callback=None)

Applies the given XBee profile to the XBee.

### **Parameters**

- **profile\_path** (String) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

### **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

#### close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

### comm\_iface

Returns the hardware interface associated to the XBee.

Returns Hardware interface of the XBee.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

### classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

#### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

**Returns** XBee object created.

Return type XBeeDevice

Raises SerialException – If the port to open does not exist or is already opened.

See also:

XBeeDevice

# del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) - The callback to delete.

<sup>&</sup>quot;port" -> Port number.

<sup>&</sup>quot;bitSize" -> Bit size.

<sup>&</sup>quot;stopBits" -> Stop bits.

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

<sup>&</sup>quot;timeout" for -> Timeout for synchronous operations (in seconds).

#### del fs frame received callback (callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_io\_sample\_received\_callback (callback)

Deletes a callback for the callback list of IOSampleReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

Parameters callback (Function) - The callback to delete.

#### del modem status received callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

Parameters callback (Function) - The callback to delete.

### del\_packet\_received\_callback(callback)

Deletes a callback for the callback list of *PacketReceived* event.

**Parameters** callback (Function) – The callback to delete.

## del\_route\_received\_callback(callback)

Deletes a callback for the callback list of RouteReceived event.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeDevice.add_route_received_callback()
```

### del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

### del socket state received callback (callback)

Deletes a callback for the callback list of SocketStateReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_user\_data\_relay\_received\_callback(callback)

Deletes a callback for the callback list of RelayDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

## **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

## disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

## enable\_bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method AbstractXBeeDevice.update\_bluetooth\_password().

Note that your XBee must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## execute\_command (parameter, value=None, apply=None)

Executes the provided command.

# **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

• ATCommandException - If response is not as expected.

#### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()

flush_queues()
Flushes the packets queue.

get_64bit_addr()
Returns the 64-bit address of the XBee.

Return type XBee64BitAddress
See also:
```

XBee64BitAddress

# get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use  $AbstractXBeeDevice.set\_io\_configuration()$  and IOMode.ADC.

**Parameters** io\_line (IOLine) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
set_io_configuration()
```

#### get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*APIOutputMode* 

## get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

### get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

### **Return type** String

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

Return type Integer

### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
IOValue
set_io_configuration()
```

## get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

**Raises** FileSystemNotSupportedException – If the XBee does not support filesystem.

### get\_firmware\_version()

Returns the firmware version of the XBee.

```
Returns Firmware version of the XBee.
```

**Return type** Bytearray

## get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

**HardwareVersion** 

### get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

**Returns** IO mode of the IO line provided.

Return type IOMode

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
set_io_configuration()
```

# get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
set_io_sampling_rate()
get_next_frame_id()
     Returns the next frame ID of the XBee.
          Returns The next frame ID of the XBee.
          Return type Integer
get_node_id()
     Returns the node identifier ('NI') value of the XBee.
          Returns Node identifier ('NI') of the XBee.
          Return type String
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
get power level()
     Returns the power level of the XBee.
          Returns Power level of the XBee.
          Return type PowerLevel
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
     See also:
     PowerLevel
     set_power_level()
get_protocol()
     Returns the current protocol of the XBee.
          Returns Current protocol of the XBee.
          Return type XBeeProtocol
     See also:
     XBeeProtocol
```

### get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

### See also:

*IOLine* 

### get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

# get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

# For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

### **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

### **Returns**

Tuple containing route data:

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

Return type Tuple

### get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

## get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

### has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_packets()
```

## has\_packets()

Returns if there are pending packets to read. This does not include explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_explicit_packets()
```

### is\_apply\_changes\_enabled()

Returns whether apply changes flag is enabled.

**Returns** *True* if apply changes flag is enabled, *False* otherwise.

Return type Boolean

## is\_open()

Returns whether this XBee is open.

**Returns** Boolean. *True* if this XBee is open, *False* otherwise.

#### is remote()

Override method.

#### See also:

```
AbstractXBeeDevice.is_remote()
```

### log

Returns the XBee logger.

**Returns** The XBee device logger.

Return type Logger

## open (force\_settings=False)

Opens the communication with the XBee and loads information about it.

**Parameters force\_settings** (Boolean, optional, default=`False`) – *True* to open the device ensuring/forcing that the specified serial settings are applied even if the current configuration is different, *False* to open the device with the current configuration.

### **Raises**

- TimeoutException If there is any problem with the communication.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the XBee is already opened.

# operating\_mode

Returns the operating mode of this XBee.

**Returns** OperatingMode. This XBee operating mode.

### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

```
read_device_info (init=True, fire_event=True)
```

Updates all instance parameters reading them from the XBee.

#### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if True.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

```
Return type IOSample
```

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
IOSample
```

#### reset()

Override method.

### See also:

```
AbstractXBeeDevice.reset()
```

#### scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

### send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

## Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

### See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

## send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

### send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns None.

#### **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

#### Returns

Response packet if sync is True, None otherwise.

```
Return type XBeePacket
```

## Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

### See also:

#### XBeePacket

# send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

**Returns** Received response packet.

Return type XBeePacket

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

#### See also:

XBeePacket

## send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

#### **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

### See also:

XBeeLocalInterface

### serial\_port

Returns the serial port associated to the XBee, if any.

### Returns

Serial port of the XBee. *None* if the local XBee does not use serial communication.

#### Return type XBeeSerialPort

#### See also:

XBeeSerialPort

#### set 16bit addr(value)

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

### set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

*APIOutputMode* 

# $\verb"set_api_output_mode_value" (api_output_mode)$

Sets the API output mode of the XBee.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputModeBit* 

## set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*IOLine* 

### set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

## **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOLine IOValue* 

### set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

#### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

### set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_io_sampling_rate()
```

### set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

Parameters node\_id (String) - New node identifier ('NI') of the XBee.

### Raises

- ValueError If *node id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

 $\texttt{set\_parameter}(parameter, value, apply=None)$ 

Override.

**See:** AbstractXBeeDevice.set\_parameter()

set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

# $\verb|set_pwm_duty_cycle| (io\_line, cycle)|$

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

#### See also:

```
IOLine
IOMode.PWM
```

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

### update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters device** (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_} & \textit{file=None, bootloader\_} & \textit{firmware\_} & \textit{file=None, bootloader\_} & \textit{file=None, bootlo$ 

Performs a firmware update operation of the XBee.

## **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- progress\_callback (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

### write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code> to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

Bases: digi.xbee.devices.IPDevice

This class represents a local Cellular device.

Class constructor. Instantiates a new CellularDevice with the provided parameters.

#### **Parameters**

- port (String) Serial port identifier. Depends on operating system. e.g. '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
- baud\_rate (Integer) Serial port baud rate.
- (Integer, default (\_sync\_ops\_timeout) serial.EIGHTBITS): Port bitsize.
- (Integer, default serial.STOPBITS\_ONE): Port stop bits.
- (Character, default (parity) serial.PARITY\_NONE): Port parity.
- (Integer, default FlowControl. NONE): Port flow control.
- (Integer, default 3): Read timeout (in seconds).
- $\bullet \ \ \textbf{comm\_iface} \ \, (\textit{XBeeCommunicationInterface}) Communication \ \, interface. \\$

Raises All exceptions raised by XBeeDevice.\_\_init\_\_() constructor.

See also:

```
XBeeDevice
XBeeDevice.__init__()
```

```
open (force_settings=False)
     Override.
     See also:
     XBeeDevice.open()
get_protocol()
     Override.
     See also:
     XBeeDevice.get_protocol()
is_device_info_complete()
     Override.
     See also:
     AbstractXBeeDevice.is device info complete()
is_connected()
     Returns whether the device is connected to the Internet.
          Returns True if connected to the Internet, False otherwise.
          Return type Boolean
          Raises
                • TimeoutException – If there is a timeout getting the association indication status.
                • XBeeException – If there is any other XBee related exception.
get_cellular_ai_status()
     Returns the current association status of this Cellular device.
     It indicates occurrences of errors during the modem initialization and connection.
          Returns
              The association indication status of the Cellular device.
          Return type CellularAssociationIndicationStatus
          Raises
                • TimeoutException – If there is a timeout getting the association indication status.
                • XBeeException – If there is any other XBee related exception.
```

## add\_sms\_callback (callback)

Adds a callback for the event SMSReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The data received as an SMSMessage

#### del sms callback (callback)

Deletes a callback for the callback list of SMSReceived event.

**Parameters** callback (Function) – The callback to delete.

### get\_imei\_addr()

Returns the IMEI address of this Cellular device.

To refresh this value use the method CellularDevice.read device info().

**Returns** The IMEI address of this Cellular device.

Return type XBeeIMEIAddress

## send\_sms (phone\_number, data)

Sends the provided SMS message to the given phone number.

This method blocks until a success or error response arrives or the configured receive timeout expires.

For non-blocking operations use the method CellularDevice.send\_sms\_async().

### **Parameters**

- **phone\_number** (String) The phone number to send the SMS to.
- data (String) Text of the SMS.

### **Raises**

- ValueError If phone\_number or data is None.
- OperationNotSupportedException If the device is remote.
- TimeoutException If there is a timeout sending the SMS.
- XBeeException If there is any other XBee related exception.

## send\_sms\_async (phone\_number, data)

Sends asynchronously the provided SMS to the given phone number.

Asynchronous transmissions do not wait for answer or for transmit status packet.

### **Parameters**

- phone\_number (String) The phone number to send the SMS to.
- data (String) Text of the SMS.

#### Raises

- ValueError If phone\_number or data is None.
- OperationNotSupportedException If the device is remote.
- XBeeException If there is any other XBee related exception.

## get\_sockets\_list()

Returns a list with the IDs of all active (open) sockets.

#### **Returns**

**list with the IDs of all active (open) sockets, or empty list** if there is not any active socket.

Return type List

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

### get\_socket\_info(socket\_id)

Returns the information of the socket with the given socket ID.

**Parameters** socket\_id (Integer) - ID of the socket.

### **Returns**

The socket information, or *None* if the socket with that ID does not exist.

Return type SocketInfo

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

#### See also:

SocketInfo

## get\_64bit\_addr()

Deprecated.

Cellular protocol does not have an associated 64-bit address.

### add\_io\_sample\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# ${\tt del\_io\_sample\_received\_callback}\ (callback)$

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# set\_dio\_change\_detection(io\_lines\_set)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# get\_io\_sampling\_rate()

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### set\_io\_sampling\_rate(rate)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### get\_node\_id()

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### set\_node\_id (node\_id)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## get\_power\_level()

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### set\_power\_level (power\_level)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

### add\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# add\_expl\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# $\verb"add_fs_frame_received_callback" (callback)$

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

### See also:

AbstractXBeeDevice FSCmd ReceiveOptions

### add\_ip\_data\_received\_callback (callback)

Adds a callback for the event IPDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The data received as an IPMessage

### add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

### add\_modem\_status\_received\_callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

### add\_packet\_received\_callback(callback)

Adds a callback for the event PacketReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

### add\_route\_received\_callback (callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

**Parameters callback** (Function) – The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

See also:

XBeeDevice.del\_route\_received\_callback()

### add\_socket\_data\_received\_callback(callback)

Adds a callback for the event SocketDataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

### add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters callback** (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

### add\_socket\_state\_received\_callback(callback)

Adds a callback for the event SocketStateReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

• The socket ID as an Integer.

• The state received as a SocketState.

### add\_user\_data\_relay\_received\_callback (callback)

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

### apply\_changes()

Applies changes via 'AC' command.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

### **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) Maximum time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

### **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

#### close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

### comm iface

Returns the hardware interface associated to the XBee.

Returns Hardware interface of the XBee.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

#### classmethod create xbee device (comm port data)

Creates and returns an XBeeDevice from data of the port to which is connected.

### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

"baudRate" -> Baud rate.

"port" -> Port number.

"bitSize" -> Bit size.

"stopBits" -> Stop bits.

"parity" -> Parity.

"flowControl" -> Flow control.

"timeout" for -> Timeout for synchronous operations (in seconds).

**Returns** XBee object created.

Return type XBeeDevice

**Raises** SerialException – If the port to open does not exist or is already opened.

See also:

XBeeDevice

# del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) - The callback to delete.

### del\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# ${\tt del\_expl\_data\_received\_callback}\ (callback)$

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### del\_fs\_frame\_received\_callback (callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_ip\_data\_received\_callback(callback)

Deletes a callback for the callback list of IPDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_modem\_status\_received\_callback (callback)

Deletes a callback for the callback list of ModemStatusReceived event.

Parameters callback (Function) - The callback to delete.

### del\_packet\_received\_callback (callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_route\_received\_callback(callback)

Deletes a callback for the callback list of RouteReceived event.

**Parameters** callback (Function) – The callback to delete.

#### See also:

XBeeDevice.add route received callback()

## del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

Parameters callback (Function) - The callback to delete.

### del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of SocketStateReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_user\_data\_relay\_received\_callback(callback)

Deletes a callback for the callback list of RelayDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

#### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

#### disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method AbstractXBeeDevice.update bluetooth password().

Note that your XBee must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

execute\_command (parameter, value=None, apply=None)

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()
```

## flush\_queues()

Flushes the packets queue.

```
get_16bit_addr()
```

Deprecated.

This protocol does not have an associated 16-bit address.

```
get_adc_value(io_line)
```

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set io configuration()</code> and <code>IOMode.ADC</code>.

Parameters io\_line (IOLine) - IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

#### See also:

```
IOLine
set_io_configuration()
```

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

Returns API output mode of the XBee.

Return type APIOutputMode

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

**APIOutputMode** 

### get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

digi.xbee.models.mode.APIOutputModeBit

# get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

Returns The Bluetooth MAC address.

Return type String

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

Return type Integer

## get\_dest\_address()

Deprecated.

Operation not supported in this protocol. Use  $IPDevice.get\_dest\_ip\_addr()$  instead. This method raises an AttributeError.

# get\_dest\_ip\_addr()

Returns the destination IP address.

**Returns** Configured destination IP address.

Return type ipaddress. IPv4Address

### Raises

- TimeoutException If there is a timeout getting the destination IP address.
- XBeeException If there is any other XBee related exception.

#### See also:

```
ipaddress. IPv4Address
```

### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

## See also:

```
IOLine
IOValue
set_io_configuration()
```

## get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

## get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

```
Return type Bytearray
get_hardware_version()
     Returns the hardware version of the XBee.
          Returns Hardware version of the XBee.
          Return type Hardware Version
     See also:
     HardwareVersion
get_io_configuration(io_line)
     Returns the configuration of the provided IO line.
          Parameters io_line (IOLine) – IO line to get its configuration.
          Returns IO mode of the IO line provided.
          Return type IOMode
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
     See also:
     IOLine
     IOMode
     set_io_configuration()
get_ip_addr()
     Returns the IP address of this IP XBee.
     To refresh this value use the method IPDevice.read_device_info().
          Returns The IP address of this IP device.
          Return type ipaddress. IPv4Address
     See also:
     ipaddress.IPv4Address
get_network()
     Deprecated.
     This protocol does not support the network functionality.
```

682

```
get_next_frame_id()
     Returns the next frame ID of the XBee.
          Returns The next frame ID of the XBee.
          Return type Integer
get pan id()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
get_pwm_duty_cycle (io_line)
     Returns the PWM duty cycle in % corresponding to the provided IO line.
          Parameters io_line (IOLine) – IO line to get its PWM duty cycle.
          Returns PWM duty cycle of the given IO line.
          Return type Integer
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
                • ValueError - If io_line has no PWM capability.
     See also:
     IOLine
get_role()
     Gets the XBee role.
          Returns the role of the XBee.
          Return type Role
     See also:
     Role
```

### get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

# For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

### **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

# Returns

# **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

# Return type Tuple

# get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

# get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

# has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

# See also:

```
XBeeDevice.has packets()
```

```
has_packets()
     Returns if there are pending packets to read. This does not include explicit packets.
          Returns True if there are pending packets, False otherwise.
          Return type Boolean
     See also:
     XBeeDevice.has_explicit_packets()
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is open()
     Returns whether this XBee is open.
          Returns Boolean. True if this XBee is open, False otherwise.
is remote()
     Override method.
     See also:
     AbstractXBeeDevice.is_remote()
log
     Returns the XBee logger.
          Returns The XBee device logger.
          Return type Logger
operating_mode
     Returns the operating mode of this XBee.
          Returns OperatingMode. This XBee operating mode.
reachable
     Returns whether the XBee is reachable.
          Returns True if the device is reachable, False otherwise.
          Return type Boolean
read_data(timeout=None)
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
read_data_from (remote_xbee, timeout=None)
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
```

### read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

#### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if True.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

Returns IO sample read from the XBee.

Return type IOSample

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*IOSample* 

# read\_ip\_data(timeout=3)

Reads new IP data received by this XBee during the provided timeout.

This method blocks until new IP data is received or the provided timeout expires.

For non-blocking operations, register a callback and use the method <code>IPDevice.add\_ip\_data\_received\_callback()</code>.

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop listening() to stop listening for incoming IP data.

**Parameters timeout** (Integer, optional) - The time to wait for new IP data in seconds.

**Returns** IP message, *None* if this device did not receive new data.

Return type IPMessage

**Raises** ValueError – If *timeout* is less than 0.

# read\_ip\_data\_from(ip\_addr, timeout=3)

Reads new IP data received from the given IP address during the provided timeout.

This method blocks until new IP data from the provided IP address is received or the given timeout expires.

For non-blocking operations, register a callback and use the method *IPDevice*. add\_ip\_data\_received\_callback().

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop\_listening() to stop listening for incoming IP data.

#### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to read data from.
- timeout (Integer, optional) The time to wait for new IP data in seconds.

#### Returns

**IP** message, *None* if this device did not receive new data from the provided IP address.

Return type IPMessage

Raises ValueError – If timeout is less than 0.

# reset()

Override method.

See also:

```
AbstractXBeeDevice.reset()
```

# scan\_counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

### send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

Parameters data (Bytearray) - Data to send.

#### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

# See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

send\_data (remote\_xbee, data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

send\_data\_async (remote\_xbee, data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

send\_data\_broadcast (data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

send\_ip\_data (ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port using the specified IP protocol. For TCP and TCP SSL protocols, you can also indicate if the socket should be closed when data is sent.

This method blocks until a success or error response arrives or the configured receive timeout expires.

#### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- **protocol** (*IPProtocol*) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- **close\_socket** (Boolean, optional, default=`False`) True to close the socket just after the transmission. False to keep it open.

# **Raises**

- ValueError If *ip\_addr* or *protocol* or *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- OperationNotSupportedException If the XBee is remote.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

# send\_ip\_data\_async (ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port asynchronously using the specified IP protocol. For TCP and TCP SSL protocols, you can also indicate if the socket should be closed when data is sent.

Asynchronous transmissions do not wait for answer from the remote device or for transmit status packet.

## **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- protocol (IPProtocol) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- **close\_socket** (Boolean, optional, default=`False`) True to close the socket just after the transmission. False to keep it open.

### **Raises**

- ValueError If ip\_addr or protocol or data is None or dest\_port is less than 0 or greater than 65535.
- OperationNotSupportedException If the XBee is remote.
- XBeeException If there is any other XBee related exception.

# send\_ip\_data\_broadcast (dest\_port, data)

Sends the provided IP data to all clients.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

#### **Parameters**

- **dest\_port** (*Integer*) The destination port of the transmission.
- data (String or Bytearray) The IP data to be sent.

### **Raises**

- ValueError If *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

# send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

## See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

# send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

## **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

#### Returns

Response packet if sync is True, None otherwise.

Return type XBeePacket

## **Raises**

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

#### See also:

XBeePacket

# send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

#### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

Returns Received response packet.

Return type XBeePacket

#### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBeePacket

# send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

### **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

# **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

### See also:

XBeeLocalInterface

### serial port

Returns the serial port associated to the XBee, if any.

#### Returns

Serial port of the XBee. None if the local XBee does not use serial communication.

Return type XBeeSerialPort

#### See also:

XBeeSerialPort

#### set 16bit addr(value)

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

# $\verb"set_api_output_mode" (api_output_mode")$

Deprecated since version 1.3: Use set api output mode value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• OperationNotSupportedException — If it is not supported by the current protocol.

#### See also:

*APIOutputMode* 

# set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

APIOutputModeBit

# set\_dest\_address(addr)

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.set\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

# set\_dest\_ip\_addr (address)

Sets the destination IP address.

Parameters address (ipaddress. IPv4Address) - Destination IP address.

## **Raises**

- ValueError If address is None.
- TimeoutException If there is a timeout setting the destination IP address.
- XBeeException If there is any other XBee related exception.

## See also:

ipaddress.IPv4Address

### set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

#### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOValue
```

# set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
IOLine
IOMode
get_io_configuration()

set_pan_id(value)
Deprecated.
Operation not supported in this protocol. This method raises an AttributeError.
set_parameter(parameter, value, apply=None)
Override.
```

**See:** AbstractXBeeDevice.set\_parameter()

### set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

#### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

# **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

#### See also:

```
IOLine
IOMode.PWM
```

# set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

### start\_listening(src\_port)

Starts listening for incoming IP transmissions in the provided port.

**Parameters** src\_port (Integer) – Port to listen for incoming transmissions.

### **Raises**

- ValueError If *source\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout setting the source port.
- XBeeException If there is any other XBee related exception.

# stop\_listening()

Stops listening for incoming IP transmissions.

#### Raises

- TimeoutException If there is a timeout processing the operation.
- XBeeException If there is any other XBee related exception.

# update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_} & (xml\_firmware\_file, xbee\_firmware\_file=None, bootloader\_firmware\_file=None, timeout=None, progress\_callback=None) \\ \end{tabular}$ 

Performs a firmware update operation of the XBee.

### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

## **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

### write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code>

to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

This class provides common functionality for XBee Low-Power Wide-Area Network devices.

Class constructor. Instantiates a new LPWANDevice with the provided parameters.

#### **Parameters**

- **port** (*String*) Serial port identifier. Depends on operating system. e.g. '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
- baud\_rate (Integer) Serial port baud rate.
- (Integer, default (\_sync\_ops\_timeout) serial.EIGHTBITS): Port bitsize.
- (Integer, default serial.STOPBITS\_ONE): Port stop bits.
- (Character, default (parity) serial.PARITY\_NONE): Port parity.
- (Integer, default FlowControl.NONE): Port flow control.
- (Integer, default 3): Read timeout (in seconds).
- comm\_iface (XBeeCommunicationInterface) Communication interface.

Raises All exceptions raised by XBeeDevice.\_\_init\_\_() constructor.

#### See also:

```
CellularDevice
CellularDevice.__init__()
```

send\_ip\_data (ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port using the specified IP protocol.

This method blocks until a success or error response arrives or the configured receive timeout expires.

# **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.

- protocol (*IPProtocol*) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- close\_socket (Boolean, optional) Must be False.

Raises ValueError – If protocol is not UDP.

# send\_ip\_data\_async (ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port asynchronously using the specified IP protocol.

Asynchronous transmissions do not wait for answer from the remote device or for transmit status packet.

### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- protocol (IPProtocol) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- close\_socket (Boolean, optional) Must be False.

**Raises** ValueError – If *protocol* is not UDP.

# add\_sms\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## del sms callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# send\_sms (phone\_number, data)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# send\_sms\_async (phone\_number, data)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# add\_bluetooth\_data\_received\_callback (callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

# add\_data\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an  ${\tt AttributeError}$ .

## add\_expl\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# add\_fs\_frame\_received\_callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (*AbstractXBeeDevice*): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

#### See also:

AbstractXBeeDevice FSCmd ReceiveOptions

## add\_io\_sample\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# add\_ip\_data\_received\_callback (callback)

Adds a callback for the event IPDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The data received as an IPMessage

# add\_micropython\_data\_received\_callback (callback)

Adds a callback for the event MicroPythonDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

# add\_modem\_status\_received\_callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

# add\_packet\_received\_callback(callback)

Adds a callback for the event PacketReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

# add\_route\_received\_callback(callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

**Parameters** callback (Function) – The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

# See also:

XBeeDevice.del\_route\_received\_callback()

#### add socket data received callback (callback)

Adds a callback for the event Socket DataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

# add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

# add\_socket\_state\_received\_callback(callback)

Adds a callback for the event SocketStateReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a *SocketState*.

# add\_user\_data\_relay\_received\_callback(callback)

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

# apply\_changes()

Applies changes via 'AC' command.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# $\verb"apply_profile" (profile_path, timeout=None, progress_callback=None)"$

Applies the given XBee profile to the XBee.

### **Parameters**

- **profile\_path** (String) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

### **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

## close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

#### comm iface

Returns the hardware interface associated to the XBee.

**Returns** Hardware interface of the XBee.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

# classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

#### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

"timeout" for -> Timeout for synchronous operations (in seconds).

**Returns** XBee object created.

Return type XBeeDevice

Raises SerialException – If the port to open does not exist or is already opened.

See also:

*XBeeDevice* 

# del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) - The callback to delete.

<sup>&</sup>quot;port" -> Port number.

<sup>&</sup>quot;bitSize" -> Bit size.

<sup>&</sup>quot;stopBits" -> Stop bits.

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

#### del data received callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# del\_expl\_data\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

Parameters callback (Function) - The callback to delete.

# del\_io\_sample\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# del\_ip\_data\_received\_callback(callback)

Deletes a callback for the callback list of IPDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del modem status received callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

Parameters callback (Function) - The callback to delete.

# del\_packet\_received\_callback(callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_route\_received\_callback(callback)

Deletes a callback for the callback list of RouteReceived event.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeDevice.add_route_received_callback()
```

# del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

Parameters callback (Function) – The callback to delete.

# del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of SocketStateReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_user\_data\_relay\_received\_callback(callback)

Deletes a callback for the callback list of RelayDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

#### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

# disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

#### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method  $AbstractXBeeDevice.update\_bluetooth\_password()$ .

Note that your XBee must include Bluetooth Low Energy support.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### execute\_command (parameter, value=None, apply=None)

Executes the provided command.

# **Parameters**

(String or (parameter) - class: .ATStringCommand): AT command to execute.

- value (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()
```

## flush\_queues()

Flushes the packets queue.

# get\_16bit\_addr()

Deprecated.

This protocol does not have an associated 16-bit address.

### get\_64bit\_addr()

Deprecated.

Cellular protocol does not have an associated 64-bit address.

# get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC</code>.

**Parameters** io\_line (*IOLine*) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• OperationNotSupportedException – If response does not contain the value for the given IO line.

## See also:

```
IOLine
set_io_configuration()
```

### get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*APIOutputMode* 

# get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

**Return type** Bytearray

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

digi.xbee.models.mode.APIOutputModeBit

# get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

Return type String

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### get\_cellular\_ai\_status()

Returns the current association status of this Cellular device.

It indicates occurrences of errors during the modem initialization and connection.

### Returns

**The association** indication status of the Cellular device.

Return type CellularAssociationIndicationStatus

# Raises

- TimeoutException If there is a timeout getting the association indication status.
- XBeeException If there is any other XBee related exception.

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

**Return type** Integer

# ${\tt get\_dest\_address}\,(\,)$

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.get\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

## get\_dest\_ip\_addr()

Returns the destination IP address.

Returns Configured destination IP address.

Return type ipaddress. IPv4Address

#### Raises

- TimeoutException If there is a timeout getting the destination IP address.
- XBeeException If there is any other XBee related exception.

See also:

```
ipaddress.IPv4Address
```

# get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

#### See also:

```
IOLine
IOValue
set_io_configuration()
```

# get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

 $\label{lem:continuous} \textbf{Raises} \ \mbox{\tt FileSystemNotSupportedException} - \mbox{\tt If the XBee does not support filesystem}.$ 

# get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

**Return type** Bytearray

# get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

```
HardwareVersion
```

```
get_imei_addr()
     Returns the IMEI address of this Cellular device.
     To refresh this value use the method CellularDevice.read device info().
          Returns The IMEI address of this Cellular device.
          Return type XBeeIMEIAddress
get_io_configuration(io_line)
     Returns the configuration of the provided IO line.
          Parameters io_line (IOLine) – IO line to get its configuration.
          Returns IO mode of the IO line provided.
          Return type IOMode
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException – If response is not as expected.
     See also:
     IOLine
     IOMode
     set_io_configuration()
get_io_sampling_rate()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_ip_addr()
     Returns the IP address of this IP XBee.
     To refresh this value use the method IPDevice.read_device_info().
          Returns The IP address of this IP device.
          Return type ipaddress. IPv4Address
```

This protocol does not support the network functionality.

See also:

get\_network() Deprecated.

ipaddress.IPv4Address

```
get_next_frame_id()
     Returns the next frame ID of the XBee.
          Returns The next frame ID of the XBee.
          Return type Integer
get node id()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_pan_id()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
get_power_level()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_protocol()
     Override.
     See also:
     XBeeDevice.get_protocol()
get_pwm_duty_cycle (io_line)
     Returns the PWM duty cycle in % corresponding to the provided IO line.
          Parameters io_line (IOLine) – IO line to get its PWM duty cycle.
          Returns PWM duty cycle of the given IO line.
          Return type Integer
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
                • ValueError – If io_line has no PWM capability.
```

See also:

```
IOLine
```

# get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

# get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

# For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

#### **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

### **Returns**

# **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

Return type Tuple

## get\_socket\_info(socket\_id)

Returns the information of the socket with the given socket ID.

**Parameters** socket\_id (Integer) - ID of the socket.

#### Returns

The socket information, or None if the socket with that ID does not exist.

Return type SocketInfo

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

#### See also:

SocketInfo

# get\_sockets\_list()

Returns a list with the IDs of all active (open) sockets.

#### Returns

list with the IDs of all active (open) sockets, or empty list if there is not any active socket

# Return type List

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

# get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

# get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

## has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

# See also:

XBeeDevice.has\_packets()

```
has_packets()
     Returns if there are pending packets to read. This does not include explicit packets.
          Returns True if there are pending packets, False otherwise.
          Return type Boolean
     See also:
     XBeeDevice.has_explicit_packets()
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is connected()
     Returns whether the device is connected to the Internet.
          Returns True if connected to the Internet, False otherwise.
          Return type Boolean
          Raises
                • TimeoutException – If there is a timeout getting the association indication status.
                • XBeeException – If there is any other XBee related exception.
is_device_info_complete()
     Override.
     See also:
     AbstractXBeeDevice.is_device_info_complete()
is_open()
     Returns whether this XBee is open.
          Returns Boolean. True if this XBee is open, False otherwise.
is remote()
     Override method.
     See also:
     AbstractXBeeDevice.is_remote()
log
     Returns the XBee logger.
          Returns The XBee device logger.
          Return type Logger
```

## open (force\_settings=False)

Override.

See also:

XBeeDevice.open()

# operating\_mode

Returns the operating mode of this XBee.

**Returns** OperatingMode. This XBee operating mode.

#### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

## read\_data(timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# read\_data\_from (remote\_xbee, timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

# **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if *True*.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

# read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

# Return type IOSample

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOSample* 

# read\_ip\_data(timeout=3)

Reads new IP data received by this XBee during the provided timeout.

This method blocks until new IP data is received or the provided timeout expires.

For non-blocking operations, register a callback and use the method IPDevice.  $add\_ip\_data\_received\_callback()$ .

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop listening() to stop listening for incoming IP data.

Parameters timeout (Integer, optional) - The time to wait for new IP data in seconds.

**Returns** IP message, *None* if this device did not receive new data.

Return type IPMessage

Raises ValueError – If timeout is less than 0.

# read\_ip\_data\_from(ip\_addr, timeout=3)

Reads new IP data received from the given IP address during the provided timeout.

This method blocks until new IP data from the provided IP address is received or the given timeout expires.

For non-blocking operations, register a callback and use the method IPDevice.  $add\_ip\_data\_received\_callback()$ .

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop\_listening() to stop listening for incoming IP data.

#### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to read data from.
- timeout (Integer, optional) The time to wait for new IP data in seconds.

#### Returns

**IP** message, *None* if this device did not receive new data from the provided IP address.

Return type IPMessage

Raises ValueError - If timeout is less than 0.

#### reset()

Override method.

#### See also:

```
AbstractXBeeDevice.reset()
```

# scan\_counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

# send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

### See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

# send\_data (remote\_xbee, data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

```
send_data_async (remote_xbee, data, transmit_options=0)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

```
send_data_broadcast (data, transmit_options=0)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# $\verb"send_ip_data_broadcast" (\textit{dest\_port}, \textit{data})$

Sends the provided IP data to all clients.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

## **Parameters**

- $dest\_port$  (Integer) The destination port of the transmission.
- data (String or Bytearray) The IP data to be sent.

### Raises

- ValueError If *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

## send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (Bytearray) – Data to send.

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

### send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

# **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

### **Returns**

Response packet if sync is True, None otherwise.

```
Return type XBeePacket
```

## Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

#### See also:

XBeePacket

### send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

**Returns** Received response packet.

Return type XBeePacket

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

#### See also:

XBeePacket

## send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

### **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If local interface is None.
- XBeeException If there is any problem sending the User Data Relay.

### See also:

XBeeLocalInterface

### serial\_port

Returns the serial port associated to the XBee, if any.

#### Returns

Serial port of the XBee. None if the local XBee does not use serial communication.

Return type XBeeSerialPort

### See also:

#### XBeeSerialPort

# set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

**Parameters value** (XBee16BitAddress) – New 16-bit address of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

# set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

# See also:

APIOutputMode

#### set api output mode value (api output mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• OperationNotSupportedException — If it is not supported by the current protocol.

#### See also:

*APIOutputModeBit* 

# set\_dest\_address(addr)

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.set\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

## set\_dest\_ip\_addr (address)

Sets the destination IP address.

Parameters address (ipaddress. IPv4Address) - Destination IP address.

#### Raises

- ValueError If address is None.
- TimeoutException If there is a timeout setting the destination IP address.
- XBeeException If there is any other XBee related exception.

#### See also:

ipaddress. IPv4Address

# set\_dio\_change\_detection(io\_lines\_set)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

*IOLine IOValue* 

### set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

#### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

## set\_io\_sampling\_rate(rate)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# set\_node\_id (node\_id)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# set\_pan\_id(value)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## set\_parameter (parameter, value, apply=None)

Override.

```
See: AbstractXBeeDevice.set_parameter()
```

## set\_power\_level (power\_level)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

# **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or cycle is not between 0 and 100.

### See also:

IOLine IOMode.PWM

# set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) – Read timeout in seconds.

### start\_listening(src\_port)

Starts listening for incoming IP transmissions in the provided port.

**Parameters** src\_port (Integer) – Port to listen for incoming transmissions.

#### Raises

- ValueError If *source\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout setting the source port.
- XBeeException If there is any other XBee related exception.

## stop\_listening()

Stops listening for incoming IP transmissions.

### Raises

- TimeoutException If there is a timeout processing the operation.
- XBeeException If there is any other XBee related exception.

### update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters device** (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

## write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
class digi.xbee.devices.NBIoTDevice(port=None,
                                                                             baud rate=None,
                                             data_bits=<sphinx.ext.autodoc.importer._MockObject
                                            object>, stop bits=<sphinx.ext.autodoc.importer. MockObject
                                             object>, parity=<sphinx.ext.autodoc.importer._MockObject
                                            object>, flow_control=<FlowControl.NONE: None>,
                                             _sync_ops_timeout=4, comm_iface=None)
     Bases: digi.xbee.devices.LPWANDevice
     This class represents a local NB-IoT device.
     Class constructor. Instantiates a new NBIOTDevice with the provided parameters.
          Parameters
                 • port (String) - Serial port identifier. Depends on operating system.
                                                                                         e.g.
                   '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
                 • baud_rate (Integer) - Serial port baud rate.
                 • (Integer, default (_sync_ops_timeout) - serial.EIGHTBITS): Port
                  bitsize.
                 • (Integer, default - serial.STOPBITS_ONE): Port stop bits.
                 • (Character, default (parity) - serial.PARITY_NONE): Port parity.
                 • (Integer, default - FlowControl.NONE): Port flow control.
                 • (Integer, default – 3): Read timeout (in seconds).
                 • comm_iface (XBeeCommunicationInterface) - Communication interface.
          Raises All exceptions raised by XBeeDevice.__init__() constructor.
     See also:
     LPWANDevice
     LPWANDevice.__init__()
     open (force_settings=False)
          Override.
          See also:
          XBeeDevice.open()
     get_protocol()
          Override.
          See also:
          XBeeDevice.get_protocol()
     add_bluetooth_data_received_callback(callback)
```

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

# add\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# add\_expl\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### add\_fs\_frame\_received\_callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters callback** (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

#### See also:

AbstractXBeeDevice FSCmd ReceiveOptions

## add\_io\_sample\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## add\_ip\_data\_received\_callback (callback)

Adds a callback for the event IPDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The data received as an IPMessage

# add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

# add\_modem\_status\_received\_callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

## add\_packet\_received\_callback (callback)

Adds a callback for the event PacketReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

### add\_route\_received\_callback (callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

**Parameters** callback (Function) – The callback. Receives three arguments.

- source (*XBeeDevice*): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

### See also:

```
XBeeDevice.del_route_received_callback()
```

#### add sms callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### add socket data received callback(callback)

Adds a callback for the event SocketDataReceived.

**Parameters callback** (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- · The data received as Bytearray.

# add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

# $\verb"add_socket_state_received_callback" (callback)$

Adds a callback for the event SocketStateReceived.

**Parameters callback** (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a SocketState.

# $\verb|add_user_data_relay_received_callback| (callback)$

Adds a callback for the event RelayDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The relay data as a *UserDataRelayMessage*.

## apply\_changes()

Applies changes via 'AC' command.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

#### **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

# close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

### comm\_iface

Returns the hardware interface associated to the XBee.

Returns Hardware interface of the XBee.

Return type XBeeCommunicationInterface

#### See also:

XBeeCommunicationInterface

# classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

#### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

```
"baudRate" -> Baud rate.
```

<sup>&</sup>quot;port" -> Port number.

<sup>&</sup>quot;bitSize" -> Bit size.

```
"stopBits" -> Stop bits.
```

"timeout" for -> Timeout for synchronous operations (in seconds).

**Returns** XBee object created.

Return type XBeeDevice

**Raises** SerialException – If the port to open does not exist or is already opened.

See also:

XBeeDevice

## del\_bluetooth\_data\_received\_callback (callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) - The callback to delete.

# del\_data\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### del\_expl\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

Parameters callback (Function) - The callback to delete.

### del\_io\_sample\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# del\_ip\_data\_received\_callback (callback)

Deletes a callback for the callback list of IPDataReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_modem\_status\_received\_callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_packet\_received\_callback (callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_route\_received\_callback (callback)

Deletes a callback for the callback list of RouteReceived event.

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

**Parameters callback** (Function) – The callback to delete.

#### See also:

```
XBeeDevice.add route received callback()
```

### del sms callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

# del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of SocketStateReceived event.

**Parameters** callback (Function) – The callback to delete.

## del\_user\_data\_relay\_received\_callback (callback)

Deletes a callback for the callback list of RelayDataReceived event.

Parameters callback (Function) - The callback to delete.

# determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (Bytearray) Firmware version to get its protocol.

## Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

### disable bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

#### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method <code>AbstractXBeeDevice.update\_bluetooth\_password()</code>.

Note that your XBee must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
execute_command(parameter, value=None, apply=None)
```

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- value (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()
```

### flush\_queues()

Flushes the packets queue.

## get\_16bit\_addr()

Deprecated.

This protocol does not have an associated 16-bit address.

### get\_64bit\_addr()

Deprecated.

Cellular protocol does not have an associated 64-bit address.

## get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC</code>.

Parameters io\_line (IOLine) - IO line to get its ADC value.

Returns Analog value corresponding to the provided IO line.

Return type Integer

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
set_io_configuration()
```

# get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

Returns API output mode of the XBee.

Return type APIOutputMode

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

#### *APIOutputMode*

# get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

Returns the parameter value.

**Return type** Bytearray

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

# get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

Returns The Bluetooth MAC address.

**Return type** String

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_cellular\_ai\_status()

Returns the current association status of this Cellular device.

It indicates occurrences of errors during the modem initialization and connection.

#### Returns

The association indication status of the Cellular device.

Return type CellularAssociationIndicationStatus

Raises

- TimeoutException If there is a timeout getting the association indication status.
- XBeeException If there is any other XBee related exception.

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

**Return type** Integer

## get\_dest\_address()

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.get\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

# get\_dest\_ip\_addr()

Returns the destination IP address.

**Returns** Configured destination IP address.

Return type ipaddress. IPv4Address

#### Raises

- TimeoutException If there is a timeout getting the destination IP address.
- XBeeException If there is any other XBee related exception.

#### See also:

ipaddress. IPv4Address

# get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

See also:

*IOLine* 

```
IOValue
     set_io_configuration()
get_file_manager()
     Returns the file system manager for the XBee.
          Returns The file system manager.
          Return type FileSystemManager
          Raises FileSystemNotSupportedException - If the XBee does not support filesys-
get firmware version()
     Returns the firmware version of the XBee.
          Returns Firmware version of the XBee.
          Return type Bytearray
get_hardware_version()
     Returns the hardware version of the XBee.
          Returns Hardware version of the XBee.
          Return type Hardware Version
     See also:
     HardwareVersion
get imei addr()
     Returns the IMEI address of this Cellular device.
     To refresh this value use the method CellularDevice.read_device_info().
          Returns The IMEI address of this Cellular device.
          Return type XBeeIMEIAddress
get_io_configuration(io_line)
     Returns the configuration of the provided IO line.
          Parameters io_line (IOLine) – IO line to get its configuration.
          Returns IO mode of the IO line provided.
          Return type IOMode
          Raises
```

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

See also:

```
IOLine
     IOMode
     set_io_configuration()
get_io_sampling_rate()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_ip_addr()
     Returns the IP address of this IP XBee.
     To refresh this value use the method IPDevice.read_device_info().
          Returns The IP address of this IP device.
          Return type ipaddress. IPv4Address
     See also:
     ipaddress.IPv4Address
get_network()
     Deprecated.
     This protocol does not support the network functionality.
get_next_frame_id()
     Returns the next frame ID of the XBee.
          Returns The next frame ID of the XBee.
          Return type Integer
get_node_id()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_pan_id()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
get_power_level()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_pwm_duty_cycle (io_line)
     Returns the PWM duty cycle in % corresponding to the provided IO line.
```

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

**Returns** PWM duty cycle of the given IO line.

Return type Integer

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

### See also:

*IOLine* 

## get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

## get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

# For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

### **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

# Returns

# **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):

- source (RemoteXBeeDevice): The source node of the route.
- destination (RemoteXBeeDevice): The destination node of the route.
- hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

# Return type Tuple

## get\_socket\_info(socket\_id)

Returns the information of the socket with the given socket ID.

**Parameters** socket\_id (Integer) - ID of the socket.

#### Returns

The socket information, or *None* if the socket with that ID does not exist.

Return type SocketInfo

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

#### See also:

SocketInfo

# get\_sockets\_list()

Returns a list with the IDs of all active (open) sockets.

#### Returns

**list with the IDs of all active (open) sockets, or empty list** if there is not any active socket.

# Return type List

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

### get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

```
get_xbee_device_callbacks()
```

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

# has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_packets()
```

## has\_packets()

Returns if there are pending packets to read. This does not include explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

See also:

```
XBeeDevice.has_explicit_packets()
```

# $\verb|is_apply_changes_enabled|()$

Returns whether apply changes flag is enabled.

**Returns** *True* if apply changes flag is enabled, *False* otherwise.

Return type Boolean

# is\_connected()

Returns whether the device is connected to the Internet.

**Returns** *True* if connected to the Internet, *False* otherwise.

Return type Boolean

**Raises** 

- TimeoutException If there is a timeout getting the association indication status.
- XBeeException If there is any other XBee related exception.

## is\_device\_info\_complete()

Override.

See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

### is\_open()

Returns whether this XBee is open.

**Returns** Boolean. *True* if this XBee is open, *False* otherwise.

### is remote()

Override method.

See also:

```
AbstractXBeeDevice.is_remote()
```

# log

Returns the XBee logger.

**Returns** The XBee device logger.

Return type Logger

### operating\_mode

Returns the operating mode of this XBee.

**Returns** OperatingMode. This XBee operating mode.

### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

# read\_data(timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# read\_data\_from(remote\_xbee, timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if True.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

See also:

```
AbstractXBeeDevice.is device info complete()
```

# read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

Return type IOSample

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOSample* 

## read\_ip\_data(timeout=3)

Reads new IP data received by this XBee during the provided timeout.

This method blocks until new IP data is received or the provided timeout expires.

For non-blocking operations, register a callback and use the method IPDevice.  $add\_ip\_data\_received\_callback()$ .

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop\_listening() to stop listening for incoming IP data.

**Parameters timeout** (Integer, optional) – The time to wait for new IP data in seconds.

**Returns** IP message, *None* if this device did not receive new data.

Return type IPMessage

**Raises** ValueError – If *timeout* is less than 0.

## read\_ip\_data\_from(ip\_addr, timeout=3)

Reads new IP data received from the given IP address during the provided timeout.

This method blocks until new IP data from the provided IP address is received or the given timeout expires.

For non-blocking operations, register a callback and use the method IPDevice. add\_ip\_data\_received\_callback().

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop\_listening() to stop listening for incoming IP data.

#### **Parameters**

•  $ip\_addr$  (ipaddress.IPv4Address) – The IP address to read data from.

• timeout (Integer, optional) - The time to wait for new IP data in seconds.

### Returns

**IP** message, *None* if this device did not receive new data from the provided IP address.

```
Return type IPMessage
```

**Raises** ValueError – If *timeout* is less than 0.

#### reset()

Override method.

#### See also:

```
AbstractXBeeDevice.reset()
```

### scan\_counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

## send\_bluetooth\_data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send_user_data_relay()
```

### send\_data (remote\_xbee, data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

```
\verb"send_data_async" (\textit{remote\_xbee}, \textit{data}, \textit{transmit\_options} = 0)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### send\_data\_broadcast (data, transmit\_options=0)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### send\_ip\_data(ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port using the specified IP protocol.

This method blocks until a success or error response arrives or the configured receive timeout expires.

#### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- protocol (IPProtocol) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- close socket (Boolean, optional) Must be False.

Raises ValueError – If protocol is not UDP.

### send\_ip\_data\_async(ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port asynchronously using the specified IP protocol.

Asynchronous transmissions do not wait for answer from the remote device or for transmit status packet.

#### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- protocol (IPProtocol) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- close\_socket (Boolean, optional) Must be False.

Raises ValueError – If protocol is not UDP.

## send\_ip\_data\_broadcast (dest\_port, data)

Sends the provided IP data to all clients.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

### **Parameters**

- **dest\_port** (*Integer*) The destination port of the transmission.
- data (String or Bytearray) The IP data to be sent.

# Raises

- ValueError If *data* is *None* or *dest port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

### send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

# See also:

XBeeDevice.send bluetooth data()

```
XBeeDevice.send user data relay()
```

# send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

### **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

#### Returns

**Response packet if sync is True, None** otherwise.

Return type XBeePacket

#### Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

#### See also:

XBeePacket

# send\_packet\_sync\_and\_get\_response(packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

#### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

Returns Received response packet.

Return type XBeePacket

# Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.

• XBeeException – If the XBee's communication interface is closed.

#### See also:

XBeePacket

### send sms (phone number, data)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# send\_sms\_async (phone\_number, data)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

#### **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

#### See also:

*XBeeLocalInterface* 

# serial\_port

Returns the serial port associated to the XBee, if any.

#### Returns

Serial port of the XBee. *None* if the local XBee does not use serial communication.

Return type XBeeSerialPort

See also:

XBeeSerialPort

# set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

## set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputMode* 

## set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

# See also:

*APIOutputModeBit* 

### set\_dest\_address(addr)

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.set\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

### set\_dest\_ip\_addr(address)

Sets the destination IP address.

Parameters address (ipaddress. IPv4Address) - Destination IP address.

#### Raises

- ValueError If address is None.
- TimeoutException If there is a timeout setting the destination IP address.
- XBeeException If there is any other XBee related exception.

### See also:

ipaddress. IPv4Address

## set\_dio\_change\_detection (io\_lines\_set)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

# **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

*IOLine IOValue* 

# set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

#### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

# set\_io\_sampling\_rate(rate)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

```
set_node_id (node_id)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

set\_pan\_id(value)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

set\_parameter (parameter, value, apply=None)

Override.

**See:** AbstractXBeeDevice.set parameter()

```
set_power_level (power_level)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- **cycle** (*Integer*) Duty cycle in % to be assigned. Must be between 0 and 100.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

 ValueError – If the given IO line does not have PWM capability or cycle is not between 0 and 100.

#### See also:

```
IOLine
IOMode.PWM
```

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

## start\_listening(src\_port)

Starts listening for incoming IP transmissions in the provided port.

**Parameters** src\_port (Integer) – Port to listen for incoming transmissions.

### **Raises**

- ValueError If *source\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout setting the source port.
- XBeeException If there is any other XBee related exception.

### stop\_listening()

Stops listening for incoming IP transmissions.

#### Raises

- TimeoutException If there is a timeout processing the operation.
- XBeeException If there is any other XBee related exception.

## update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### update device data from (device)

Updates the current node information with provided data. This is only for internal use.

Parameters device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

Performs a firmware update operation of the XBee.

#### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

# write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
class digi.xbee.devices.WiFiDevice (port=None,
                                                                              baud rate=None,
                                           data_bits=<sphinx.ext.autodoc.importer._MockObject ob-
                                           ject>, stop bits=<sphinx.ext.autodoc.importer. MockObject
                                           object>, parity=<sphinx.ext.autodoc.importer._MockObject
                                           object>, flow_control=<FlowControl.NONE: None>,
                                            _sync_ops_timeout=4, comm_iface=None)
     Bases: digi.xbee.devices.IPDevice
     This class represents a local Wi-Fi XBee.
     Class constructor. Instantiates a new WiFiDevice with the provided parameters.
          Parameters
                 • port (String) - Serial port identifier. Depends on operating system.
                                                                                          e.g.
                   '/dev/ttyUSB0' on 'GNU/Linux' or 'COM3' on Windows.
                 • baud_rate (Integer) - Serial port baud rate.
                 • (Integer, default (_sync_ops_timeout) - serial.EIGHTBITS): Port
                  bitsize.
                 • (Integer, default - serial.STOPBITS_ONE): Port stop bits.
                 • (Character, default (parity) - serial.PARITY_NONE): Port parity.
                 • (Integer, default - FlowControl.NONE): Port flow control.
                 • (Integer, default – 3): Read timeout (in seconds).
                 • comm_iface (XBeeCommunicationInterface) - Communication interface.
          Raises All exceptions raised by XBeeDevice.__init__() constructor.
     See also:
     IPDevice
     v.__init__()
     open (force settings=False)
          Override.
          See also:
          XBeeDevice.open()
     get_protocol()
          Override.
          See also:
          XBeeDevice.get_protocol()
     get_wifi_ai_status()
```

Returns the current association status of the device.

### **Returns**

Current association status of the device.

Return type WiFiAssociationIndicationStatus

### Raises

- TimeoutException If there is a timeout getting the association indication status.
- XBeeException If there is any other XBee related exception.

### See also:

WiFiAssociationIndicationStatus

# get\_access\_point (ssid)

Finds and returns the access point that matches the supplied SSID.

**Parameters** ssid (String) – SSID of the access point to get.

### Returns

**Discovered access point with the provided** SID, or *None* if the timeout expires and the access point was not found.

Return type AccessPoint

## Raises

- TimeoutException If there is a timeout getting the access point.
- XBeeException If there is an error sending the discovery command.

### See also:

AccessPoint

# scan\_access\_points()

Performs a scan to search for access points in the vicinity.

This method blocks until all the access points are discovered or the configured access point timeout expires.

The access point timeout is configured using the WiFiDevice.set\_access\_point\_timeout() method and can be consulted with WiFiDevice.get\_access\_point\_timeout() method.

**Returns** List of *AccessPoint* objects discovered.

Return type List

## Raises

- TimeoutException If there is a timeout scanning the access points.
- XBeeException If there is any other XBee related exception.

# See also:

AccessPoint

```
connect_by_ap (access_point, password=None)
```

Connects to the provided access point.

This method blocks until the connection with the access point is established or the configured access point timeout expires.

The access point timeout is configured using the <code>WiFiDevice.set\_access\_point\_timeout()</code> method and can be consulted with <code>WiFiDevice.get\_access\_point\_timeout()</code> method.

Once the module is connected to the access point, you can issue the <code>WiFiDevice.write\_changes()</code> method to save the connection settings. This way the module will try to connect to the access point every time it is powered on.

#### **Parameters**

- access\_point (AccessPoint) The access point to connect to.
- **password** (*String*, *optional*) The password for the access point, *None* if it does not have any encryption enabled.

### **Returns**

*True* if the module connected to the access point successfully, *False* otherwise.

# Return type Boolean

### **Raises**

- ValueError If access\_point is None.
- TimeoutException If there is a timeout sending the connect commands.
- XBeeException If there is any other XBee related exception.

# See also:

```
WiFiDevice.connect_by_ssid()
WiFiDevice.disconnect()
WiFiDevice.get_access_point()
WiFiDevice.get_access_point_timeout()
WiFiDevice.scan_access_points()
WiFiDevice.set_access_point_timeout()
```

### connect\_by\_ssid (ssid, password=None)

Connects to the access point with provided SSID.

This method blocks until the connection with the access point is established or the configured access point timeout expires.

The access point timeout is configured using the  $WiFiDevice.set\_access\_point\_timeout()$  method and can be consulted with  $WiFiDevice.get\_access\_point\_timeout()$  method.

Once the module is connected to the access point, you can issue the <code>WiFiDevice.write\_changes()</code> method to save the connection settings. This way the module will try to connect to the access point every time it is powered on.

# **Parameters**

• **ssid** (String) – SSID of the access point to connect to.

• **password** (*String*, *optional*) – The password for the access point, *None* if it does not have any encryption enabled.

#### Returns

*True* if the module connected to the access point successfully, *False* otherwise.

### Return type Boolean

#### Raises

- ValueError If ssid is None.
- TimeoutException If there is a timeout sending the connect commands.
- XBeeException If the access point with the provided SSID cannot be found.
- XBeeException If there is any other XBee related exception.

### See also:

```
WiFiDevice.connect_by_ap()
WiFiDevice.disconnect()
WiFiDevice.get_access_point()
WiFiDevice.get_access_point_timeout()
WiFiDevice.scan_access_points()
WiFiDevice.set_access_point_timeout()
```

#### disconnect()

Disconnects from the access point that the device is connected to.

This method blocks until the device disconnects totally from the access point or the configured access point timeout expires.

The access point timeout is configured using the WiFiDevice.set\_access\_point\_timeout() method and can be consulted with WiFiDevice.get\_access\_point\_timeout() method.

# Returns

*True* if the module disconnected from the access point successfully, *False* otherwise.

### Return type Boolean

#### Raises

- TimeoutException If there is a timeout sending the disconnect command.
- XBeeException If there is any other XBee related exception.

### See also:

```
WiFiDevice.connect_by_ap()
WiFiDevice.connect_by_ssid()
WiFiDevice.get_access_point_timeout()
WiFiDevice.set_access_point_timeout()
```

### is\_connected()

Returns whether the device is connected to an access point or not.

#### **Returns**

*True* if the device is connected to an access point, *False* otherwise.

Return type Boolean

Raises TimeoutException – If there is a timeout getting the association indication status.

#### See also:

```
WiFiDevice.get_wifi_ai_status()
WiFiAssociationIndicationStatus
```

# get\_access\_point\_timeout()

Returns the configured access point timeout for connecting, disconnecting and scanning access points.

**Returns** The current access point timeout in milliseconds.

Return type Integer

See also:

```
WiFiDevice.set_access_point_timeout()
```

### set\_access\_point\_timeout (ap\_timeout)

Configures the access point timeout in milliseconds for connecting, disconnecting and scanning access points.

**Parameters** ap\_timeout (Integer) - The new access point timeout in milliseconds.

**Raises** ValueError – If *ap\_timeout* is less than 0.

See also:

```
WiFiDevice.get_access_point_timeout()
```

### get\_ip\_addressing\_mode()

Returns the IP addressing mode of the device.

**Returns** The IP addressing mode.

Return type IPAddressingMode

Raises TimeoutException – If there is a timeout reading the IP addressing mode.

See also:

```
WiFiDevice.set_ip_addressing_mode()
IPAddressingMode
```

# $set_{ip}_addressing_mode(mode)$

Sets the IP addressing mode of the device.

```
Parameters mode (IPAddressingMode) – The new IP addressing mode to set.
```

Raises TimeoutException – If there is a timeout setting the IP addressing mode.

### See also:

```
WiFiDevice.get_ip_addressing_mode()
IPAddressingMode
```

### set\_ip\_address (ip\_address)

Sets the IP address of the module.

This method can only be called if the module is configured in IPAddressingMode.STATIC mode. Otherwise an *XBeeException* will be thrown.

Parameters ip\_address (ipaddress.IPv4Address) - New IP address to set.

Raises TimeoutException – If there is a timeout setting the IP address.

### See also:

```
WiFiDevice.get_mask_address()
ipaddress.IPv4Address
```

### get\_mask\_address()

Returns the subnet mask IP address.

Returns The subnet mask IP address.

Return type ipaddress. IPv4Address

Raises TimeoutException - If there is a timeout reading the subnet mask address.

#### See also:

```
WiFiDevice.set_mask_address()
ipaddress.IPv4Address
```

### set\_mask\_address (mask\_address)

Sets the subnet mask IP address.

This method can only be called if the module is configured in IPAddressingMode.STATIC mode. Otherwise an *XBeeException* will be thrown.

Parameters mask\_address (ipaddress.IPv4Address) - New subnet mask address to set.

 $\textbf{Raises} \ \ \texttt{TimeoutException} - If \ there \ is \ a \ timeout \ setting \ the \ subnet \ mask \ address.$ 

### See also:

```
WiFiDevice.get_mask_address()
ipaddress.IPv4Address
```

get\_gateway\_address()

```
Returns the IP address of the gateway.
         Returns The IP address of the gateway.
         Return type ipaddress. IPv4Address
         Raises TimeoutException – If there is a timeout reading the gateway address.
     See also:
     WiFiDevice.set_dns_address()
     ipaddress.IPv4Address
set_gateway_address(gateway_address)
     Sets the IP address of the gateway.
     This method can only be called if the module is configured in IPAddressingMode.STATIC mode.
     Otherwise an XBeeException will be thrown.
         Parameters gateway_address (ipaddress.IPv4Address) - The new gateway ad-
             dress to set.
         Raises TimeoutException – If there is a timeout setting the gateway address.
     See also:
     WiFiDevice.get_gateway_address()
     ipaddress. IPv4Address
get_dns_address()
     Returns the IP address of Domain Name Server (DNS).
         Returns The DNS address configured.
         Return type ipaddress. IPv4Address
          Raises TimeoutException – If there is a timeout reading the DNS address.
     See also:
     WiFiDevice.set_dns_address()
     ipaddress. IPv4Address
set_dns_address (dns_address)
     Sets the IP address of Domain Name Server (DNS).
         Parameters dns_address (ipaddress.IPv4Address) - The new DNS address to
         Raises TimeoutException – If there is a timeout setting the DNS address.
     See also:
```

```
WiFiDevice.get_dns_address()
ipaddress.IPv4Address
```

### add\_bluetooth\_data\_received\_callback (callback)

Adds a callback for the event BluetoothDataReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The Bluetooth data as a Bytearray.

### add\_data\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## add\_expl\_data\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### add\_fs\_frame\_received\_callback(callback)

Adds a callback for the event FileSystemFrameReceived.

**Parameters** callback (Function) – The callback. Receives four arguments.

- Source (AbstractXBeeDevice): The node that sent the file system frame.
- Frame id (Integer): The received frame id.
- Command (FSCmd): The file system command.
- Receive options (Integer): Bitfield indicating receive options.

#### See also:

```
AbstractXBeeDevice
FSCmd
ReceiveOptions
```

### add\_io\_sample\_received\_callback(callback)

Adds a callback for the event IOSampleReceived.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The received IO sample as an *IOSample*.
- The remote XBee which sent the packet as a RemoteXBeeDevice.
- The time in which the packet was received as an Integer.

# $\verb"add_ip_data_received_callback" (callback)$

Adds a callback for the event IPDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The data received as an IPMessage

# $\verb"add_micropython_data_received_callback" (callback)$

Adds a callback for the event MicroPythonDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The MicroPython data as a Bytearray.

#### add modem status received callback(callback)

Adds a callback for the event ModemStatusReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The modem status as a ModemStatus.

### add\_packet\_received\_callback (callback)

Adds a callback for the event PacketReceived.

**Parameters** callback (Function) – The callback. Receives one argument.

• The received packet as a XBeeAPIPacket.

### add\_route\_received\_callback (callback)

Adds a callback for the event RouteReceived. This works for Zigbee and Digimesh devices.

**Parameters callback** (Function) – The callback. Receives three arguments.

- source (XBeeDevice): The source node.
- destination (RemoteXBeeDevice): The destination node.
- hops (List): List of intermediate hops from closest to source to closest to destination (RemoteXBeeDevice).

#### See also:

```
XBeeDevice.del route received callback()
```

## add\_socket\_data\_received\_callback(callback)

Adds a callback for the event SocketDataReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The data received as Bytearray.

# add\_socket\_data\_received\_from\_callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The socket ID as an Integer.
- Source address pair (host, port) where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The data received as Bytearray.

# add\_socket\_state\_received\_callback(callback)

Adds a callback for the event SocketStateReceived.

**Parameters callback** (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a SocketState.

# $\verb|add_user_data_relay_received_callback| (callback)$

Adds a callback for the event RelayDataReceived.

**Parameters callback** (Function) – The callback. Receives one argument.

• The relay data as a UserDataRelayMessage.

## apply\_changes()

Applies changes via 'AC' command.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

### **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) Maximum time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

### Raises

- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

## close()

Closes the communication with the XBee.

This method guarantees that all threads running are stopped and the serial port is closed.

### comm iface

Returns the hardware interface associated to the XBee.

Returns Hardware interface of the XBee.

Return type XBeeCommunicationInterface

### See also:

XBeeCommunicationInterface

# classmethod create\_xbee\_device(comm\_port\_data)

Creates and returns an XBeeDevice from data of the port to which is connected.

#### **Parameters**

- comm\_port\_data (Dictionary) Dictionary with all comm port data needed.
- dictionary keys are (The) -

"baudRate" -> Baud rate.

"port" -> Port number.

"bitSize" -> Bit size.

"stopBits" -> Stop bits.

"parity" -> Parity.

"flowControl" -> Flow control.

"timeout" for -> Timeout for synchronous operations (in seconds).

**Returns** XBee object created.

Return type XBeeDevice

**Raises** SerialException – If the port to open does not exist or is already opened.

See also:

XBeeDevice

### del\_bluetooth\_data\_received\_callback(callback)

Deletes a callback for the callback list of BluetoothDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_data\_received\_callback (callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### del\_expl\_data\_received\_callback(callback)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

# del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters callback** (Function) – The callback to delete.

### del\_io\_sample\_received\_callback (callback)

Deletes a callback for the callback list of IOSampleReceived event.

**Parameters** callback (Function) – The callback to delete.

# del\_ip\_data\_received\_callback (callback)

Deletes a callback for the callback list of IPDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_modem\_status\_received\_callback (callback)

Deletes a callback for the callback list of ModemStatusReceived event.

Parameters callback (Function) - The callback to delete.

### del\_packet\_received\_callback (callback)

Deletes a callback for the callback list of PacketReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_route\_received\_callback(callback)

Deletes a callback for the callback list of RouteReceived event.

**Parameters** callback (Function) – The callback to delete.

#### See also:

XBeeDevice.add\_route\_received\_callback()

# del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

**Parameters** callback (Function) – The callback to delete.

### del\_socket\_state\_received\_callback(callback)

Deletes a callback for the callback list of SocketStateReceived event.

**Parameters** callback (Function) – The callback to delete.

### del\_user\_data\_relay\_received\_callback (callback)

Deletes a callback for the callback list of RelayDataReceived event.

**Parameters** callback (Function) – The callback to delete.

### determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

#### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

### disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
enable_apply_changes (value)
```

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method AbstractXBeeDevice.update bluetooth password().

Note that your XBee must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

execute\_command (parameter, value=None, apply=None)

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()
```

## flush\_queues()

Flushes the packets queue.

```
get_16bit_addr()
```

Deprecated.

This protocol does not have an associated 16-bit address.

### get\_64bit\_addr()

Returns the 64-bit address of the XBee.

**Returns** 64-bit address of the XBee.

Return type XBee64BitAddress

See also:

XBee64BitAddress

# get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC</code>.

Parameters io\_line (IOLine) - IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
    set_io_configuration()

get_api_output_mode()
```

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*APIOutputMode* 

### get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

digi.xbee.models.mode.APIOutputModeBit

## get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

Return type String

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

**Return type** Integer

## get\_dest\_address()

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.get\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

# get\_dest\_ip\_addr()

Returns the destination IP address.

Returns Configured destination IP address.

Return type ipaddress. IPv4Address

### Raises

- TimeoutException If there is a timeout getting the destination IP address.
- XBeeException If there is any other XBee related exception.

#### See also:

```
ipaddress. IPv4Address
```

### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

# Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

#### See also:

```
IOLine
IOValue
set_io_configuration()
```

### get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

# get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

**Return type** Bytearray

### get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

HardwareVersion

### get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

Returns IO mode of the IO line provided.

Return type IOMode

**Raises** 

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
IOLine
IOMode
set_io_configuration()
```

#### get io sampling rate()

Returns the IO sampling rate of the XBee.

Returns IO sampling rate of XBee.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
set_io_sampling_rate()
get_ip_addr()
     Returns the IP address of this IP XBee.
     To refresh this value use the method IPDevice.read device info().
          Returns The IP address of this IP device.
          Return type ipaddress. IPv4Address
     See also:
     ipaddress.IPv4Address
get_network()
     Deprecated.
     This protocol does not support the network functionality.
get_next_frame_id()
     Returns the next frame ID of the XBee.
          Returns The next frame ID of the XBee.
          Return type Integer
get_node_id()
     Returns the node identifier ('NI') value of the XBee.
          Returns Node identifier ('NI') of the XBee.
          Return type String
get_pan_id()
     Deprecated.
     Operation not supported in this protocol. This method raises an AttributeError.
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
```

```
get_power_level()
```

Returns the power level of the XBee.

**Returns** Power level of the XBee.

Return type PowerLevel

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
PowerLevel
set_power_level()
```

### get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) - IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

# See also:

*IOLine* 

# get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

### get\_route\_to\_node (remote, timeout=10, force=True)

Gets the route from this XBee to the given remote node.

### For Zigbee:

- 'AR' parameter of the local node must be configured with a value different from 'FF'.
- Set *force* to *True* to force the Zigbee remote node to return its route independently of the local node configuration as high or low RAM concentrator ('DO' of the local value)

### **Parameters**

- remote (RemoteXBeeDevice) The remote node.
- timeout (Float, optional, default=10) Maximum number of seconds to wait for the route.
- **force** (Boolean) *True* to force asking for the route, *False* otherwise. Only for Zigbee.

### **Returns**

### **Tuple containing route data:**

- status (*TransmitStatus*): The transmit status.
- Tuple with route data (*None* if the route was not read in the provided timeout):
  - source (RemoteXBeeDevice): The source node of the route.
  - destination (RemoteXBeeDevice): The destination node of the route.
  - hops (List): List of intermediate nodes (RemoteXBeeDevice) ordered from closest to source to closest to destination node (source and destination not included).

# Return type Tuple

# get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

## get\_xbee\_device\_callbacks()

Returns this XBee internal callbacks for process received packets.

This method is called by the PacketListener associated with this XBee to get its callbacks. These callbacks are executed before user callbacks.

Returns PacketReceived

# has\_explicit\_packets()

Returns if there are pending explicit packets to read. This does not include non-explicit packets.

**Returns** *True* if there are pending packets, *False* otherwise.

Return type Boolean

# See also:

```
XBeeDevice.has packets()
```

```
has_packets()
     Returns if there are pending packets to read. This does not include explicit packets.
          Returns True if there are pending packets, False otherwise.
          Return type Boolean
     See also:
     XBeeDevice.has_explicit_packets()
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is_device_info_complete()
     Override.
     See also:
     AbstractXBeeDevice.is device info complete()
is_open()
     Returns whether this XBee is open.
          Returns Boolean. True if this XBee is open, False otherwise.
is_remote()
     Override method.
     See also:
     AbstractXBeeDevice.is_remote()
log
     Returns the XBee logger.
          Returns The XBee device logger.
          Return type Logger
operating_mode
     Returns the operating mode of this XBee.
          Returns OperatingMode. This XBee operating mode.
reachable
     Returns whether the XBee is reachable.
          Returns True if the device is reachable, False otherwise.
          Return type Boolean
```

768

#### read data(timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

### read\_data\_from(remote\_xbee, timeout=None)

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

## read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if *True*.
- **fire\_event** (Boolean, optional, default=`True`) *True* to throw and update event if any parameter changed, *False* otherwise.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

Return type IOSample

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*IOSample* 

```
read ip data(timeout=3)
```

Reads new IP data received by this XBee during the provided timeout.

This method blocks until new IP data is received or the provided timeout expires.

For non-blocking operations, register a callback and use the method *IPDevice*. add\_ip\_data\_received\_callback().

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method <code>IPDevice.start\_listening()</code> for that purpose. When finished, you can use the method <code>IPDevice.stop\_listening()</code> to stop listening for incoming IP data.

Parameters timeout (Integer, optional) - The time to wait for new IP data in seconds.

**Returns** IP message, *None* if this device did not receive new data.

Return type IPMessage

**Raises** ValueError – If timeout is less than 0.

```
read_ip_data_from(ip_addr, timeout=3)
```

Reads new IP data received from the given IP address during the provided timeout.

This method blocks until new IP data from the provided IP address is received or the given timeout expires.

For non-blocking operations, register a callback and use the method IPDevice add\_ip\_data\_received\_callback().

Before reading IP data you need to start listening for incoming IP data at a specific port. Use the method IPDevice.start\_listening() for that purpose. When finished, you can use the method IPDevice.stop\_listening() to stop listening for incoming IP data.

# **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to read data from.
- timeout (Integer, optional) The time to wait for new IP data in seconds.

### **Returns**

**IP** message, *None* if this device did not receive new data from the provided IP address.

Return type IPMessage

**Raises** ValueError – If timeout is less than 0.

#### reset()

Override method.

See also:

```
AbstractXBeeDevice.reset()
```

#### scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

### send bluetooth data(data)

Sends the given data to the Bluetooth interface using a User Data Relay frame.

**Parameters data** (*Bytearray*) – Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

#### See also:

```
XBeeDevice.send_micropython_data()
XBeeDevice.send user data relay()
```

```
send_data (remote_xbee, data, transmit_options=0)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

```
\verb"send_data_async" (\textit{remote\_xbee}, \textit{data}, \textit{transmit\_options} = 0)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

```
send_data_broadcast (data, transmit_options=0)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

```
send_ip_data (ip_addr, dest_port, protocol, data, close_socket=False)
```

Sends the provided IP data to the given IP address and port using the specified IP protocol. For TCP and TCP SSL protocols, you can also indicate if the socket should be closed when data is sent.

This method blocks until a success or error response arrives or the configured receive timeout expires.

### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- **protocol** (*IPProtocol*) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- **close\_socket** (Boolean, optional, default=`False`) True to close the socket just after the transmission. False to keep it open.

#### Raises

- ValueError If *ip\_addr* or *protocol* or *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- OperationNotSupportedException If the XBee is remote.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

### send\_ip\_data\_async (ip\_addr, dest\_port, protocol, data, close\_socket=False)

Sends the provided IP data to the given IP address and port asynchronously using the specified IP protocol. For TCP and TCP SSL protocols, you can also indicate if the socket should be closed when data is sent.

Asynchronous transmissions do not wait for answer from the remote device or for transmit status packet.

### **Parameters**

- ip\_addr (ipaddress.IPv4Address) The IP address to send IP data to.
- **dest\_port** (*Integer*) The destination port of the transmission.
- protocol (IPProtocol) The IP protocol used for the transmission.
- data (String or Bytearray) The IP data to be sent.
- **close\_socket** (Boolean, optional, default=`False`) True to close the socket just after the transmission. False to keep it open.

### **Raises**

- ValueError If *ip\_addr* or *protocol* or *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- OperationNotSupportedException If the XBee is remote.
- XBeeException If there is any other XBee related exception.

## send\_ip\_data\_broadcast (dest\_port, data)

Sends the provided IP data to all clients.

This method blocks until a success or error transmit status arrives or the configured receive timeout expires.

#### **Parameters**

- **dest\_port** (*Integer*) The destination port of the transmission.
- data (String or Bytearray) The IP data to be sent.

### **Raises**

- ValueError If *data* is *None* or *dest\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout sending the data.
- XBeeException If there is any other XBee related exception.

# send\_micropython\_data(data)

Sends the given data to the MicroPython interface using a User Data Relay frame.

**Parameters data** (Bytearray) – Data to send.

### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If there is any problem sending the data.

### See also:

```
XBeeDevice.send_bluetooth_data()
XBeeDevice.send_user_data_relay()
```

### send\_packet (packet, sync=False)

Sends the packet and waits for the response. The packet to send is escaped depending on the current operating mode.

This method can be synchronous or asynchronous.

If synchronous, this method discards all response packets until it finds the one that has the appropriate frame ID, that is, the sent packet's frame ID.

If asynchronous, this method does not wait for any response and returns *None*.

#### **Parameters**

- packet (XBeePacket) The packet to send.
- **sync** (Boolean) *True* to wait for the response of the sent packet and return it, *False* otherwise.

### **Returns**

Response packet if sync is True, None otherwise.

Return type XBeePacket

#### Raises

- TimeoutException If *sync* is *True* and the response packet for the sent one cannot be read.
- InvalidOperatingModeException If the XBee operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException If the packet listener is not running or the XBee's communication interface is closed.

#### See also:

XBeePacket

# send\_packet\_sync\_and\_get\_response (packet\_to\_send, timeout=None)

Sends the packet and waits for its corresponding response.

### **Parameters**

- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer, optional, default=`None`) Number of seconds to wait. -1 to wait indefinitely.

**Returns** Received response packet.

Return type XBeePacket

### **Raises**

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If response is not received in the configured timeout.
- XBeeException If the XBee's communication interface is closed.

### See also:

XBeePacket

### send\_user\_data\_relay (local\_interface, data)

Sends the given data to the given XBee local interface.

#### **Parameters**

- local\_interface (XBeeLocalInterface) Destination XBee local interface.
- data (Bytearray) Data to send.

#### Raises

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ValueError If *local\_interface* is *None*.
- XBeeException If there is any problem sending the User Data Relay.

### See also:

XBeeLocalInterface

### serial\_port

Returns the serial port associated to the XBee, if any.

#### Returns

**Serial port of the XBee.** *None* if the local XBee does not use serial communication.

Return type XBeeSerialPort

See also:

XBeeSerialPort

## set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

**Parameters value** (XBee16BitAddress) – New 16-bit address of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

### set\_api\_output\_mode (api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

 $\textbf{Parameters api\_output\_mode} \ (\textit{APIOutputMode}) - New \ API \ output \ mode.$ 

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputMode* 

### set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

# See also:

*APIOutputModeBit* 

### set dest address(addr)

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.set\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

### set\_dest\_ip\_addr(address)

Sets the destination IP address.

Parameters address (ipaddress.IPv4Address) - Destination IP address.

# Raises

- ValueError If address is None.
- TimeoutException If there is a timeout setting the destination IP address.

• XBeeException – If there is any other XBee related exception.

#### See also:

ipaddress.IPv4Address

### set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*IOLine* 

# set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

#### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*IOLine IOValue* 

# $\verb|set_io_configuration| (io\_line, io\_mode)|$

Sets the configuration of the provided IO line.

### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

### set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters rate** (Integer) – New IO sampling rate of the XBee in seconds.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_io_sampling_rate()
```

### set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

## Raises

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
set_pan_id(value)
```

Deprecated.

Operation not supported in this protocol. This method raises an AttributeError.

set\_parameter (parameter, value, apply=None)

Override.

**See:** AbstractXBeeDevice.set\_parameter()

### set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

## set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

#### See also:

```
IOLine
IOMode.PWM
```

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

# start\_listening(src\_port)

Starts listening for incoming IP transmissions in the provided port.

**Parameters** src\_port (Integer) – Port to listen for incoming transmissions.

#### Raises

- ValueError If *source\_port* is less than 0 or greater than 65535.
- TimeoutException If there is a timeout setting the source port.
- XBeeException If there is any other XBee related exception.

### stop\_listening()

Stops listening for incoming IP transmissions.

#### Raises

- TimeoutException If there is a timeout processing the operation.
- XBeeException If there is any other XBee related exception.

### update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters device** (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

Performs a firmware update operation of the XBee.

### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.

- timeout (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

#### write changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code> to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

Bases: digi.xbee.devices.AbstractXBeeDevice

This class represents a remote XBee.

Class constructor. Instantiates a new RemoteXBeeDevice with the provided parameters.

#### **Parameters**

- local\_xbee (XBeeDevice) Local XBee associated with the remote one.
- **x64bit** addr (XBee64BitAddress) 64-bit address of the remote XBee.
- x16bit\_addr (XBee16BitAddress) 16-bit address of the remote XBee.
- node\_id (String, optional) Node identifier of the remote XBee.

See also:

```
XBee16BitAddress
XBee64BitAddress
XBeeDevice
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
    See also:
    AbstractXBeeDevice.get_parameter()
set_parameter (parameter, value, apply=None)
     Override.
    See also:
    AbstractXBeeDevice.set_parameter()
is_remote()
     Override method.
    See also:
    AbstractXBeeDevice.is_remote()
reset()
     Override method.
     See also:
    AbstractXBeeDevice.reset()
get_local_xbee_device()
     Returns the local XBee associated to the remote one.
         Returns Local XBee.
         Return type XBeeDevice
set_local_xbee_device (local_xbee_device)
     This methods associates a XBeeDevice to the remote XBee.
         Parameters local_xbee_device (XBeeDevice) - New local XBee associated to the
             remote one.
    See also:
     XBeeDevice
```

### get\_serial\_port()

Returns the serial port of the local XBee associated to the remote one.

#### Returns

Serial port of the local XBee associated to the remote one.

Return type XBeeSerialPort

See also:

XBeeSerialPort

### get\_comm\_iface()

Returns the communication interface of the local XBee associated to the remote one.

### **Returns**

**Communication interface of the** local XBee associated to the remote one.

Return type XBeeCommunicationInterface

See also:

XBeeCommunicationInterface

### get\_ota\_max\_block\_size()

Returns the maximum number of bytes to send for ota updates.

Returns Maximum ota block size to send.

Return type Integer

## set\_ota\_max\_block\_size(size)

Sets the maximum number of bytes to send for ota updates.

**Parameters** size (Integer) – Maximum ota block size to send.

**Raises** ValueError – If size is not between 0 and 255.

 $\verb"update_filesystem_image" (ota\_filesystem\_file, timeout=None, progress\_callback=None)$ 

Performs a filesystem image update operation of the device.

#### **Parameters**

- ota\_filesystem\_file (String) Location of the OTA filesystem image file.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the update process.
- **progress\_callback** (Function, optional) Function to receive progress information. Receives two arguments:
  - The current update task as a String.
  - The current update task percentage as an Integer.

### Raises

• XBeeException - If the device is not open.

- InvalidOperatingModeException If the device operating mode is invalid.
- FileSystemNotSupportedException If the filesystem update is not supported in the XBee.
- FileSystemException If there is any error performing the filesystem update.

#### apply\_changes()

Applies changes via 'AC' command.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

#### **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*, *default=`None`*) **Maximum** time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

# Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

# determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

## disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

### enable\_bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method <code>AbstractXBeeDevice.update\_bluetooth\_password()</code>.

Note that your XBee must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

execute\_command(parameter, value=None, apply=None)

Executes the provided command.

### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
```

```
AbstractXBeeDevice.write changes()
     AbstractXBeeDevice.is_apply_changes_enabled()
     AbstractXBeeDevice.enable_apply_changes()
get_16bit_addr()
     Returns the 16-bit address of the XBee.
          Returns 16-bit address of the XBee.
         Return type XBee16BitAddress
     See also:
     XBee16BitAddress
get_64bit_addr()
     Returns the 64-bit address of the XBee.
         Returns 64-bit address of the XBee.
         Return type XBee64BitAddress
     See also:
     XBee64BitAddress
get_adc_value(io_line)
     Returns the analog value of the provided IO line.
     The provided IO line must be previously configured as ADC. To do so, use AbstractXBeeDevice.
     set io configuration() and IOMode.ADC.
         Parameters io_line (IOLine) – IO line to get its ADC value.
         Returns Analog value corresponding to the provided IO line.
         Return type Integer
         Raises
               • TimeoutException – If response is not received before the read timeout expires.
               • XBeeException – If the XBee's communication interface is closed.
               • InvalidOperatingModeException - If the XBee's operating mode is not API
                 or ESCAPED API. This method only checks the cached value of the operating mode.
               • ATCommandException - If response is not as expected.
               • OperationNotSupportedException - If response does not contain the value
                 for the given IO line.
     See also:
     IOLine
     set_io_configuration()
```

#### get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*APIOutputMode* 

# get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

### get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

### **Return type** String

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

Return type Integer

### get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

**Returns** 64-bit address of destination XBee.

Return type XBee64BitAddress

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
XBee64BitAddress
set dest address()
```

### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• OperationNotSupportedException – If response does not contain the value for the given IO line.

#### See also:

```
IOLine
IOValue
set_io_configuration()
```

### get\_file\_manager()

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

 $\label{lem:Raises} \textbf{Raises} \ \texttt{FileSystemNotSupportedException} - If \ the \ XBee \ does \ not \ support \ filesystem.$ 

# get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

Return type Bytearray

# get\_hardware\_version()

Returns the hardware version of the XBee.

Returns Hardware version of the XBee.

Return type HardwareVersion

See also:

**HardwareVersion** 

### get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

**Returns** IO mode of the IO line provided.

Return type IOMode

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

See also:

```
IOLine
IOMode
set_io_configuration()
```

# get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

Returns IO sampling rate of XBee.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
set_io_sampling_rate()
```

### get\_node\_id()

Returns the node identifier ('NI') value of the XBee.

Returns Node identifier ('NI') of the XBee.

Return type String

# get\_pan\_id()

Returns the operating PAN ID of the XBee.

**Returns** Operating PAN ID of the XBee.

Return type Bytearray

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
set_pan_id()

get power level()
```

Returns the power level of the XBee.

**Returns** Power level of the XBee.

### Return type PowerLevel

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
set_power_level()
```

### get\_protocol()

Returns the current protocol of the XBee.

**Returns** Current protocol of the XBee.

Return type XBeeProtocol

See also:

XBeeProtocol

### get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If *io line* has no PWM capability.

### See also:

*IOLine* 

#### get\_role()

Gets the XBee role.

```
Returns the role of the XBee.
```

Return type Role

See also:

Role

### get\_sync\_ops\_timeout()

Returns the serial port read timeout.

**Returns** Serial port read timeout in seconds.

Return type Integer

# $\verb|is_apply_changes_enabled|()$

Returns whether apply changes flag is enabled.

**Returns** *True* if apply changes flag is enabled, *False* otherwise.

Return type Boolean

# is\_device\_info\_complete()

Returns whether XBee node information is complete.

**Returns** *True* if node information is complete, *False* otherwise.

Return type Boolean

See also:

```
AbstractXBeeDevice.read_device_info()
```

# log

Returns the XBee logger.

**Returns** The XBee device logger.

Return type Logger

#### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

### read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if True.
- **fire\_event** (Boolean, optional, default=`True`) *True* to throw and update event if any parameter changed, *False* otherwise.

### **Raises**

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

### read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

Returns IO sample read from the XBee.

Return type IOSample

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

### See also:

*IOSample* 

#### scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

**Return type** Integer

### set 16bit addr(value)

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

### set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputMode* 

#### set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

APIOutputModeBit

### set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

### Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

### See also:

```
XBee64BitAddress
get_dest_address()
```

# set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOLine* 

# set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

*IOLine IOValue* 

### set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

#### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

### set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_io_sampling_rate()
```

### set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

### **Raises**

- ValueError If *node id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
get_pan_id()
```

### set\_power\_level (power\_level)

Sets the power level of the XBee.

Parameters power\_level (PowerLevel) - New power level of the XBee.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

# $\verb|set_pwm_duty_cycle| (io\_line, cycle)|$

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

# Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or cycle is not between 0 and 100.

#### See also:

IOLine
IOMode.PWM

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

### update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

 $\textbf{Parameters} \ \ \textbf{device} \ (\texttt{AbstractXBeeDevice}) - \textbf{XBee} \ \ \textbf{to} \ \ \textbf{get} \ \ \textbf{the} \ \ \textbf{data} \ \ \textbf{from}.$ 

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

 $\label{lem:continuous} \begin{tabular}{ll} \textbf{update\_firmware\_file} & xbee\_firmware\_file=None, \ bootloader\_firmware\_file=None, \ timeout=None, \ progress\_callback=None) \end{tabular}$ 

Performs a firmware update operation of the XBee.

#### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.

- timeout (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- progress\_callback (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

#### write changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method <code>is\_apply\_changes\_enabled()</code> to get its status and <code>enable\_apply\_changes()</code> to enable/disable the option. Method <code>apply\_changes()</code> can be used in order to manually apply the changes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

This class represents a remote 802.15.4 XBee.

Class constructor. Instantiates a new RemoteXBeeDevice with the provided parameters.

# Parameters

- local xbee (XBeeDevice) Local XBee associated with the remote one.
- x64bit\_addr (XBee64BitAddress) 64-bit address of the remote XBee.
- x16bit\_addr (XBee16BitAddress) 16-bit address of the remote XBee.
- node\_id (String, optional) Node identifier of the remote XBee.

**Raises** XBeeException – If the protocol of *local xbee* is invalid.

See also:

```
RemoteXBeeDevice
XBee16BitAddress
XBee64BitAddress
XBeeDevice
```

# get\_protocol()

Override.

See also:

```
RemoteXBeeDevice.get_protocol()
```

### set\_64bit\_addr (address)

Sets the 64-bit address of this remote 802.15.4 device.

Parameters address (XBee 64BitAddress) - The 64-bit address to set.

**Raises** ValueError – If address is None.

# get\_ai\_status()

Returns the current association status of this XBee. It indicates occurrences of errors during the modem initialization and connection.

#### Returns

The XBee association indication status.

Return type AssociationIndicationStatus

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# apply\_changes()

Applies changes via 'AC' command.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

 $\verb"apply_profile" (profile_path, timeout=None, progress_callback=None)"$ 

Applies the given XBee profile to the XBee.

### **Parameters**

• **profile\_path** (*String*) – Path of the XBee profile file to apply.

- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

### determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

#### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

```
Return type XBeeProtocol
```

### disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

# enable\_bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method AbstractXBeeDevice.update\_bluetooth\_password().

Note that your XBee must include Bluetooth Low Energy support.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

execute\_command(parameter, value=None, apply=None)

Executes the provided command.

#### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()

get_16bit_addr()
Returns the 16-bit address of the XBee.

Return type XBee16BitAddress
See also:

XBee16BitAddress

get_64bit_addr()
Returns the 64-bit address of the XBee.
```

**Returns** 64-bit address of the XBee. **Return type** XBee64BitAddress

#### See also:

XBee64BitAddress

### get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC</code>.

**Parameters** io\_line (*IOLine*) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
set_io_configuration()
```

# get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

#### *APIOutputMode*

# get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

Returns the parameter value.

**Return type** Bytearray

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

### get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

**Return type** String

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_comm\_iface()

Returns the communication interface of the local XBee associated to the remote one.

### **Returns**

**Communication interface of the** local XBee associated to the remote one.

Return type XBeeCommunicationInterface

### See also:

XBeeCommunicationInterface

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

**Return type** Integer

### get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

Returns 64-bit address of destination XBee.

Return type XBee64BitAddress

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
XBee64BitAddress
set dest address()
```

### get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

Returns current value of the provided IO line.

Return type IOValue

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

#### See also:

*IOLine* 

```
IOValue
     set_io_configuration()
get_file_manager()
     Returns the file system manager for the XBee.
          Returns The file system manager.
          Return type FileSystemManager
          Raises FileSystemNotSupportedException - If the XBee does not support filesys-
get firmware version()
     Returns the firmware version of the XBee.
          Returns Firmware version of the XBee.
          Return type Bytearray
get_hardware_version()
     Returns the hardware version of the XBee.
          Returns Hardware version of the XBee.
          Return type Hardware Version
     See also:
     HardwareVersion
get_io_configuration(io_line)
     Returns the configuration of the provided IO line.
          Parameters io_line (IOLine) - IO line to get its configuration.
          Returns IO mode of the IO line provided.
          Return type IOMode
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
     See also:
```

2.6. API reference 805

*IOLine IOMode* 

set\_io\_configuration()

### get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
set_io_sampling_rate()
```

### get\_local\_xbee\_device()

Returns the local XBee associated to the remote one.

Returns Local XBee.

Return type XBeeDevice

### get\_node\_id()

Returns the node identifier ('NI') value of the XBee.

**Returns** Node identifier ('NI') of the XBee.

Return type String

# get\_ota\_max\_block\_size()

Returns the maximum number of bytes to send for ota updates.

**Returns** Maximum ota block size to send.

Return type Integer

### get\_pan\_id()

Returns the operating PAN ID of the XBee.

**Returns** Operating PAN ID of the XBee.

Return type Bytearray

### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
set_pan_id()

get_parameter (parameter, parameter_value=None, apply=None)
   Override.
   See also:

AbstractXBeeDevice.get_parameter()
```

### get\_power\_level()

Returns the power level of the XBee.

Returns Power level of the XBee.

Return type PowerLevel

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
PowerLevel
set_power_level()
```

# get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) - IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

### See also:

*IOLine* 

```
get_role()
     Gets the XBee role.
          Returns the role of the XBee.
          Return type Role
     See also:
     Role
get_serial_port()
     Returns the serial port of the local XBee associated to the remote one.
              Serial port of the local XBee associated to the remote one.
          Return type XBeeSerialPort
     See also:
     XBeeSerialPort
get_sync_ops_timeout()
     Returns the serial port read timeout.
          Returns Serial port read timeout in seconds.
          Return type Integer
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is_device_info_complete()
     Returns whether XBee node information is complete.
          Returns True if node information is complete, False otherwise.
          Return type Boolean
     See also:
     AbstractXBeeDevice.read_device_info()
is_remote()
     Override method.
     See also:
     AbstractXBeeDevice.is_remote()
```

### log

Returns the XBee logger.

**Returns** The XBee device logger.

Return type Logger

#### reachable

Returns whether the XBee is reachable.

**Returns** *True* if the device is reachable, *False* otherwise.

Return type Boolean

### read\_device\_info (init=True, fire\_event=True)

Updates all instance parameters reading them from the XBee.

#### **Parameters**

- init (Boolean, optional, default=`True`) If False only not initialized parameters are read, all if *True*.
- **fire\_event** (Boolean, optional, default=`True`) True to throw and update event if any parameter changed, False otherwise.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

### read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

Returns IO sample read from the XBee.

Return type IOSample

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

*IOSample* 

#### reset()

Override method.

See also:

AbstractXBeeDevice.reset()

### scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

### set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

**Parameters value** (XBee16BitAddress) – New 16-bit address of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- $\bullet$  OperationNotSupportedException If the protocol is not 802.15.4.

### set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

#### See also:

*APIOutputMode* 

### set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputModeBit* 

### set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

#### See also:

```
XBee64BitAddress
get_dest_address()
```

# $\verb|set_dio_change_detection| (io\_lines\_set)$

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

**Parameters** io\_lines\_set - Set of IOLine.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

• ATCommandException – If response is not as expected.

#### See also:

*IOLine* 

### set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

*IOLine IOValue* 

# set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

### set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
get_io_sampling_rate()
```

### set\_local\_xbee\_device (local\_xbee\_device)

This methods associates a XBeeDevice to the remote XBee.

**Parameters** local\_xbee\_device (XBeeDevice) - New local XBee associated to the remote one.

#### See also:

*XBeeDevice* 

#### set node id(node id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

# Raises

- ValueError If *node id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# set\_ota\_max\_block\_size(size)

Sets the maximum number of bytes to send for ota updates.

**Parameters** size (Integer) – Maximum ota block size to send.

**Raises** ValueError – If size is not between 0 and 255.

### set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_pan_id()
```

set\_parameter (parameter, value, apply=None)

Override.

See also:

```
AbstractXBeeDevice.set parameter()
```

### set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

### set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

### See also:

```
IOLine
IOMode.PWM
```

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

### update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

**update\_filesystem\_image** (ota\_filesystem\_file, timeout=None, progress\_callback=None)
Performs a filesystem image update operation of the device.

### **Parameters**

- ota\_filesystem\_file (String) Location of the OTA filesystem image file.
- **timeout** (Integer, optional) Maximum time to wait for target read operations during the update process.
- progress\_callback (Function, optional) Function to receive progress information. Receives two arguments:

- The current update task as a String.
- The current update task percentage as an Integer.

### **Raises**

- XBeeException If the device is not open.
- InvalidOperatingModeException If the device operating mode is invalid.
- FileSystemNotSupportedException If the filesystem update is not supported in the XBee.
- FileSystemException If there is any error performing the filesystem update.

#### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- progress\_callback (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

### write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

Bases: digi.xbee.devices.RemoteXBeeDevice

This class represents a remote DigiMesh XBee device.

Class constructor. Instantiates a new RemoteDigiMeshDevice with the provided parameters.

#### **Parameters**

- **local xbee** (*XBeeDevice*) Local XBee associated with the remote one.
- **x64bit** addr (XBee64BitAddress) 64-bit address of the remote XBee.
- node\_id(String, optional) Node identifier of the remote XBee.

**Raises** XBeeException – If the protocol of *local\_xbee* is invalid.

#### See also:

```
RemoteXBeeDevice
XBee64BitAddress
XBeeDevice
```

get\_protocol()

Override.

See also:

```
RemoteXBeeDevice.get_protocol()
```

 $\verb"get_neighbors" (neighbor\_cb=None, finished\_cb=None, timeout=None)$ 

Returns the neighbors of this XBee. If *neighbor\_cb* is not defined, the process blocks during the specified timeout.

#### **Parameters**

- neighbor\_cb (Function, optional, default=`None`) Method called when a new neighbor is received. Receives two arguments:
  - The XBee that owns this new neighbor.
  - The new neighbor.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that is searching for its neighbors.
  - A list with the discovered neighbors.

- An error message if something went wrong.
- timeout (Float, optional, default=`NeighborFinder. DEFAULT\_TIMEOUT`) The timeout in seconds.

#### Returns

**List of** *Neighbor* **when** *neighbor\_cb* **is not defined,** *None* otherwise (in this case neighbors are received in the callback).

# **Return type** List

Raises OperationNotSupportedException - If XBee protocol is not DigiMesh.

### See also:

```
com.digi.models.zdo.Neighbor
```

### apply\_changes()

Applies changes via 'AC' command.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

#### **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the apply profile (seconds).
- progress\_callback (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

### **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

# determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

• hardware\_version (Integer) – Hardware version to get its protocol.

• **firmware\_version** (Bytearray) – Firmware version to get its protocol.

#### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

# disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

#### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method AbstractXBeeDevice.update\_bluetooth\_password().

Note that your XBee must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# $\verb"execute_command" (parameter, value=None, apply=None)"$

Executes the provided command.

### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.get_parameter()
    AbstractXBeeDevice.set_parameter()
    AbstractXBeeDevice.apply_changes()
    AbstractXBeeDevice.write_changes()
    AbstractXBeeDevice.is_apply_changes_enabled()
    AbstractXBeeDevice.enable_apply_changes()
get_16bit_addr()
    Returns the 16-bit address of the XBee.
        Returns 16-bit address of the XBee.
        Return type XBeel6BitAddress
    See also:
    XBee16BitAddress
get_64bit_addr()
    Returns the 64-bit address of the XBee.
         Returns 64-bit address of the XBee.
        Return type XBee64BitAddress
    See also:
```

# get\_adc\_value(io\_line)

XBee64BitAddress

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC.</code>

**Parameters** io\_line (IOLine) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

### See also:

```
IOLine
set io configuration()
```

### get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

Returns API output mode of the XBee.

Return type APIOutputMode

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
APIOutputMode
```

### get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

Returns the parameter value.

Return type Bytearray

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

 OperationNotSupportedException – If it is not supported by the current protocol.

#### See also:

digi.xbee.models.mode.APIOutputModeBit

### get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

Return type String

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### get\_comm\_iface()

Returns the communication interface of the local XBee associated to the remote one.

### **Returns**

Communication interface of the local XBee associated to the remote one.

Return type XBeeCommunicationInterface

#### See also:

XBeeCommunicationInterface

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

**Return type** Integer

# get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

**Returns** 64-bit address of destination XBee.

Return type XBee64BitAddress

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
XBee64BitAddress
set_dest_address()
```

# get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set io configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

# See also:

```
IOLine
IOValue
set_io_configuration()
```

## get\_file\_manager()

Returns the file system manager for the XBee.

Returns The file system manager.

 $\textbf{Return type} \ \textit{FileSystemManager}$ 

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

## get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

Return type Bytearray

### get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

HardwareVersion

## get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

Returns IO mode of the IO line provided.

Return type IOMode

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
IOLine
IOMode
set_io_configuration()
```

### get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set_io_sampling_rate()
```

```
get_local_xbee_device()
     Returns the local XBee associated to the remote one.
          Returns Local XBee.
          Return type XBeeDevice
get node id()
     Returns the node identifier ('NI') value of the XBee.
          Returns Node identifier ('NI') of the XBee.
          Return type String
get_ota_max_block_size()
     Returns the maximum number of bytes to send for ota updates.
          Returns Maximum ota block size to send.
          Return type Integer
get_pan_id()
     Returns the operating PAN ID of the XBee.
          Returns Operating PAN ID of the XBee.
          Return type Bytearray
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
     See also:
     set_pan_id()
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
get_power_level()
     Returns the power level of the XBee.
          Returns Power level of the XBee.
          Return type PowerLevel
          Raises
```

2.6. API reference 825

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
PowerLevel
set power level()
```

# get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

**Returns** PWM duty cycle of the given IO line.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- ValueError If *io\_line* has no PWM capability.

### See also:

*IOLine* 

# get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

# get\_serial\_port()

Returns the serial port of the local XBee associated to the remote one.

### **Returns**

Serial port of the local XBee associated to the remote one.

Return type XBeeSerialPort

```
See also:
     XBeeSerialPort
get_sync_ops_timeout()
     Returns the serial port read timeout.
          Returns Serial port read timeout in seconds.
          Return type Integer
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is_device_info_complete()
     Returns whether XBee node information is complete.
          Returns True if node information is complete, False otherwise.
          Return type Boolean
     See also:
     AbstractXBeeDevice.read_device_info()
is remote()
     Override method.
     See also:
     AbstractXBeeDevice.is remote()
log
     Returns the XBee logger.
          Returns The XBee device logger.
          Return type Logger
reachable
     Returns whether the XBee is reachable.
          Returns True if the device is reachable, False otherwise.
          Return type Boolean
read_device_info (init=True, fire_event=True)
     Updates all instance parameters reading them from the XBee.
          Parameters
```

2.6. API reference 827

ized parameters are read, all if True.

• init (Boolean, optional, default=`True`) - If False only not initial-

• **fire\_event** (Boolean, optional, default=`True`) – True to throw and update event if any parameter changed, False otherwise.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

```
Return type IOSample
```

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

### See also:

```
IOSample
```

### reset()

Override method.

# See also:

```
AbstractXBeeDevice.reset()
```

#### scan counter

828

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

```
set 16bit addr(value)
```

Sets the 16-bit address of the XBee.

Parameters value (XBee16BitAddress) - New 16-bit address of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

## set\_api\_output\_mode (api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

*APIOutputMode* 

### set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

#### *APIOutputModeBit*

# set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

### See also:

```
XBee64BitAddress
get_dest_address()
```

## set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

# **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*IOLine* 

## set dio value(io line, io value)

Sets the digital value (high or low) to the provided IO line.

## **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

# Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

*IOLine IOValue* 

# set\_io\_configuration(io\_line, io\_mode)

Sets the configuration of the provided IO line.

### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

# set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_io_sampling_rate()
```

### set local xbee device (local xbee device)

This methods associates a XBeeDevice to the remote XBee.

**Parameters** local\_xbee\_device (XBeeDevice) - New local XBee associated to the remote one.

### See also:

XBeeDevice

### set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) - New node identifier ('NI') of the XBee.

### **Raises**

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## set\_ota\_max\_block\_size(size)

Sets the maximum number of bytes to send for ota updates.

**Parameters** size (Integer) – Maximum ota block size to send.

Raises ValueError – If size is not between 0 and 255.

# set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_pan_id()
```

## set\_parameter (parameter, value, apply=None)

Override.

#### See also:

```
AbstractXBeeDevice.set parameter()
```

# set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

## set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

## See also:

```
IOLine
IOMode.PWM
```

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

## update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

**update\_filesystem\_image** (*ota\_filesystem\_file*, *timeout=None*, *progress\_callback=None*)

Performs a filesystem image update operation of the device.

#### **Parameters**

- ota\_filesystem\_file (String) Location of the OTA filesystem image file.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the update process.
- **progress\_callback** (Function, optional) Function to receive progress information. Receives two arguments:
  - The current update task as a String.
  - The current update task percentage as an Integer.

## Raises

- XBeeException If the device is not open.
- InvalidOperatingModeException If the device operating mode is invalid.
- FileSystemNotSupportedException If the filesystem update is not supported in the XBee.
- FileSystemException If there is any error performing the filesystem update.

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_file}, & xbee\_firmware\_file=None, & bootloader\_firmware\_file=None, \\ & timeout=None, & progress\_callback=None) \\ \end{tabular}$ 

Performs a firmware update operation of the XBee.

### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.

- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

# write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

 $\begin{array}{ll} \textbf{class} \ \text{digi.xbee.devices.RemoteDigiPointDevice} \ (local\_xbee, \\ node\_id=None) \\ \textbf{Bases:} \ digi.xbee.devices.RemoteXBeeDevice} \end{array}$ 

This class represents a remote DigiPoint XBee.

Class constructor. Instantiates a new RemoteDigiMeshDevice with the provided parameters.

# **Parameters**

- local\_xbee (XBeeDevice) Local XBee associated with the remote one.
- **x64bit** addr (XBee64BitAddress) 64-bit address of the remote XBee.
- node\_id(String, optional) Node identifier of the remote XBee.

Raises XBeeException - If the protocol of local\_xbee is invalid.

```
RemoteXBeeDevice

XBee64BitAddress

XBeeDevice

get_protocol()

Override.

See also:

RemoteXBeeDevice.get_protocol()
```

# apply\_changes()

Applies changes via 'AC' command.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

## **Parameters**

- **profile path** (*String*) Path of the XBee profile file to apply.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

## Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

### determine protocol (hardware version, firmware version)

Determines the XBee protocol based on the given hardware and firmware versions.

## **Parameters**

• hardware\_version (Integer) - Hardware version to get its protocol.

• **firmware\_version** (Bytearray) – Firmware version to get its protocol.

#### Returns

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

# disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

#### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method AbstractXBeeDevice.update\_bluetooth\_password().

Note that your XBee must include Bluetooth Low Energy support.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# $\verb"execute_command" (parameter, value=None, apply=None)"$

Executes the provided command.

### **Parameters**

- (String or (parameter) class: .ATStringCommand): AT command to execute.
- **value** (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
AbstractXBeeDevice.get_parameter()
    AbstractXBeeDevice.set_parameter()
    AbstractXBeeDevice.apply_changes()
    AbstractXBeeDevice.write_changes()
    AbstractXBeeDevice.is_apply_changes_enabled()
    AbstractXBeeDevice.enable_apply_changes()
get_16bit_addr()
    Returns the 16-bit address of the XBee.
        Returns 16-bit address of the XBee.
        Return type XBeel6BitAddress
    See also:
    XBee16BitAddress
get_64bit_addr()
    Returns the 64-bit address of the XBee.
         Returns 64-bit address of the XBee.
        Return type XBee64BitAddress
```

XBee64BitAddress

# get\_adc\_value(io\_line)

See also:

Returns the analog value of the provided IO line.

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC.</code>

**Parameters** io\_line (IOLine) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

```
IOLine
set io configuration()
```

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

*APIOutputMode* 

### get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

Returns the parameter value.

Return type Bytearray

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

• OperationNotSupportedException — If it is not supported by the current protocol.

### See also:

digi.xbee.models.mode.APIOutputModeBit

## get\_bluetooth\_mac\_addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

**Returns** The Bluetooth MAC address.

Return type String

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_comm\_iface()

Returns the communication interface of the local XBee associated to the remote one.

### **Returns**

Communication interface of the local XBee associated to the remote one.

Return type XBeeCommunicationInterface

### See also:

XBeeCommunicationInterface

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

**Return type** Integer

# get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

Returns 64-bit address of destination XBee.

Return type XBee64BitAddress

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
XBee64BitAddress
set_dest_address()
```

# get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

# See also:

```
IOLine
IOValue
set_io_configuration()
```

## get\_file\_manager()

Returns the file system manager for the XBee.

Returns The file system manager.

Return type FileSystemManager

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

## get\_firmware\_version()

Returns the firmware version of the XBee.

Returns Firmware version of the XBee.

Return type Bytearray

```
get_hardware_version()
```

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type Hardware Version

See also:

HardwareVersion

## get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its configuration.

Returns IO mode of the IO line provided.

Return type IOMode

**Raises** 

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
IOLine
IOMode
set_io_configuration()
```

### get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
set_io_sampling_rate()
```

```
get_local_xbee_device()
     Returns the local XBee associated to the remote one.
          Returns Local XBee.
          Return type XBeeDevice
get node id()
     Returns the node identifier ('NI') value of the XBee.
          Returns Node identifier ('NI') of the XBee.
          Return type String
get_ota_max_block_size()
     Returns the maximum number of bytes to send for ota updates.
          Returns Maximum ota block size to send.
          Return type Integer
get_pan_id()
     Returns the operating PAN ID of the XBee.
          Returns Operating PAN ID of the XBee.
          Return type Bytearray
          Raises
                • TimeoutException – If response is not received before the read timeout expires.
                • XBeeException – If the XBee's communication interface is closed.
                • InvalidOperatingModeException - If the XBee's operating mode is not API
                  or ESCAPED API. This method only checks the cached value of the operating mode.
                • ATCommandException - If response is not as expected.
     See also:
     set_pan_id()
get_parameter (parameter, parameter_value=None, apply=None)
     Override.
     See also:
     AbstractXBeeDevice.get_parameter()
get_power_level()
     Returns the power level of the XBee.
          Returns Power level of the XBee.
          Return type PowerLevel
```

2.6. API reference 843

• TimeoutException – If response is not received before the read timeout expires.

**Raises** 

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
PowerLevel
set_power_level()
```

# get\_pwm\_duty\_cycle (io\_line)

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) – IO line to get its PWM duty cycle.

**Returns** PWM duty cycle of the given IO line.

Return type Integer

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.
- ValueError If *io\_line* has no PWM capability.

### See also:

*IOLine* 

# get\_role()

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

# get\_serial\_port()

Returns the serial port of the local XBee associated to the remote one.

### **Returns**

 $\label{eq:Serial port of the local XBee associated} \ \ \text{to the remote one.}$ 

Return type XBeeSerialPort

```
See also:
     XBeeSerialPort
get_sync_ops_timeout()
     Returns the serial port read timeout.
          Returns Serial port read timeout in seconds.
          Return type Integer
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is_device_info_complete()
     Returns whether XBee node information is complete.
          Returns True if node information is complete, False otherwise.
          Return type Boolean
     See also:
     AbstractXBeeDevice.read_device_info()
is remote()
     Override method.
     See also:
     AbstractXBeeDevice.is remote()
log
     Returns the XBee logger.
          Returns The XBee device logger.
          Return type Logger
reachable
     Returns whether the XBee is reachable.
          Returns True if the device is reachable, False otherwise.
          Return type Boolean
read_device_info (init=True, fire_event=True)
     Updates all instance parameters reading them from the XBee.
          Parameters
```

2.6. API reference 845

ized parameters are read, all if True.

• init (Boolean, optional, default=`True`) - If False only not initial-

• **fire\_event** (Boolean, optional, default=`True`) – True to throw and update event if any parameter changed, False otherwise.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

Return type IOSample

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

### See also:

```
IOSample
```

# reset()

Override method.

# See also:

```
AbstractXBeeDevice.reset()
```

#### scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

## set 16bit addr(value)

Sets the 16-bit address of the XBee.

**Parameters value** (XBee16BitAddress) – New 16-bit address of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

## set\_api\_output\_mode (api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

Parameters api\_output\_mode (APIOutputMode) - New API output mode.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

*APIOutputMode* 

### set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

#### *APIOutputModeBit*

# set\_dest\_address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

#### See also:

```
XBee64BitAddress
get_dest_address()
```

## set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io\_lines\_set - Set of IOLine.

# **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

*IOLine* 

## set dio value(io line, io value)

Sets the digital value (high or low) to the provided IO line.

## **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

# Raises

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

*IOLine IOValue* 

## set\_io\_configuration (io\_line, io\_mode)

Sets the configuration of the provided IO line.

### **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

# set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_io_sampling_rate()
```

### set local xbee device (local xbee device)

This methods associates a XBeeDevice to the remote XBee.

**Parameters** local\_xbee\_device (XBeeDevice) - New local XBee associated to the remote one.

### See also:

XBeeDevice

### set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters** node\_id (String) – New node identifier ('NI') of the XBee.

### **Raises**

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## set\_ota\_max\_block\_size(size)

Sets the maximum number of bytes to send for ota updates.

**Parameters** size (Integer) – Maximum ota block size to send.

Raises ValueError – If size is not between 0 and 255.

## set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### See also:

```
get_pan_id()
```

## set\_parameter (parameter, value, apply=None)

Override.

#### See also:

```
AbstractXBeeDevice.set parameter()
```

# set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

### See also:

```
PowerLevel
get_power_level()
```

## set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

## See also:

```
IOLine
IOMode.PWM
```

### set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

## update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

**update\_filesystem\_image** (*ota\_filesystem\_file*, *timeout=None*, *progress\_callback=None*)

Performs a filesystem image update operation of the device.

#### **Parameters**

- ota\_filesystem\_file (String) Location of the OTA filesystem image file.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the update process.
- **progress\_callback** (Function, optional) Function to receive progress information. Receives two arguments:
  - The current update task as a String.
  - The current update task percentage as an Integer.

## Raises

- XBeeException If the device is not open.
- InvalidOperatingModeException If the device operating mode is invalid.
- FileSystemNotSupportedException If the filesystem update is not supported in the XBee.
- FileSystemException If there is any error performing the filesystem update.

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_file}, & xbee\_firmware\_file=None, & bootloader\_firmware\_file=None, \\ & timeout=None, & progress\_callback=None) \\ \end{tabular}$ 

Performs a firmware update operation of the XBee.

### **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.

- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- **timeout** (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

## write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

 $\begin{array}{c} \textbf{class} \ \text{digi.xbee.devices.RemoteZigBeeDevice} (local\_xbee, & x64bit\_addr=None, \\ & x16bit\_addr=None, node\_id=None) \\ \text{Bases: } digi.xbee.devices.RemoteXBeeDevice} \end{array}$ 

This class represents a remote Zigbee XBee.

Class constructor. Instantiates a new RemoteDigiMeshDevice with the provided parameters.

## **Parameters**

- local\_xbee (XBeeDevice) Local XBee associated with the remote one.
- x64bit addr (XBee64BitAddress) 64-bit address of the remote XBee.
- x16bit addr (XBee16BitAddress) 16-bit address of the remote XBee.
- node\_id (String, optional) Node identifier of the remote XBee.

**Raises** XBeeException – If the protocol of *local\_xbee* is invalid.

## See also:

```
RemoteXBeeDevice
XBee16BitAddress
XBee64BitAddress
XBeeDevice
```

### parent

Returns the parent of the XBee if it is an end device.

### **Returns**

The parent of the node for end devices, *None* if unknown or if it is not an end device.

Return type AbstractXBeeDevice

```
get_protocol()
```

Override.

See also:

```
RemoteXBeeDevice.get protocol()
```

```
is_device_info_complete()
```

Override.

See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

# get\_ai\_status()

Returns the current association status of this XBee. It indicates occurrences of errors during the modem initialization and connection.

# Returns

The XBee association indication status.

Return type AssociationIndicationStatus

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### force disassociate()

Forces this XBee to immediately disassociate from the network and re-attempt to associate.

Only valid for End Devices.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_routes (route\_cb=None, finished\_cb=None, timeout=None)

Returns the routes of this XBee. If *route\_cb* is not defined, the process blocks until the complete routing table is read.

#### **Parameters**

- route\_cb (Function, optional, default=`None`) Method called when a new route is received. Receives two arguments:
  - The XBee that owns this new route.
  - The new route.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that executed the ZDO command.
  - A list with the discovered routes.
  - An error message if something went wrong.
- timeout (Float, optional, default=`RouteTableReader. DEFAULT\_TIMEOUT`) - The ZDO command timeout in seconds.

## **Returns**

**List of** *Route* **when** *route\_cb* **is not defined,** *None* otherwise (in this case routes are received in the callback).

# Return type List

Raises OperationNotSupportedException - If XBee protocol is not Zigbee or Smart Energy.

# See also:

```
com.digi.models.zdo.Route
```

### get\_neighbors (neighbor\_cb=None, finished\_cb=None, timeout=None)

Returns the neighbors of this XBee. If *neighbor\_cb* is not defined, the process blocks until the complete neighbor table is read.

# **Parameters**

• neighbor\_cb (Function, optional, default=`None`) - Method called when a new neighbor is received. Receives two arguments:

- The XBee that owns this new neighbor.
- The new neighbor.
- **finished\_cb** (Function, optional, default=`None`) Method to execute when the process finishes. Receives three arguments:
  - The XBee that executed the ZDO command.
  - A list with the discovered neighbors.
  - An error message if something went wrong.
- timeout (Float, optional, default=`NeighborTableReader. DEFAULT\_TIMEOUT`) The ZDO command timeout in seconds.

#### Returns

**List of** *Neighbor* **when** *neighbor\_cb* **is not defined,** *None* otherwise (in this case neighbors are received in the callback).

# **Return type** List

Raises OperationNotSupportedException - If XBee protocol is not Zigbee or Smart Energy.

### See also:

```
com.digi.models.zdo.Neighbor
```

## apply\_changes()

Applies changes via 'AC' command.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

apply\_profile (profile\_path, timeout=None, progress\_callback=None)
Applies the given XBee profile to the XBee.

### **Parameters**

- **profile\_path** (*String*) Path of the XBee profile file to apply.
- timeout (Integer, optional, default=`None`) Maximum time to wait for target read operations during the apply profile (seconds).
- **progress\_callback** (Function, optional, default=`None`) Function to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

## **Raises**

• XBeeException - If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- UpdateProfileException If there is any error applying the XBee profile.

# determine\_protocol (hardware\_version, firmware\_version)

Determines the XBee protocol based on the given hardware and firmware versions.

#### **Parameters**

- hardware\_version (Integer) Hardware version to get its protocol.
- **firmware\_version** (*Bytearray*) Firmware version to get its protocol.

### **Returns**

**XBee protocol corresponding to the given** hardware and firmware versions.

Return type XBeeProtocol

## disable\_bluetooth()

Disables the Bluetooth interface of this XBee.

Note that your device must include Bluetooth Low Energy support.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# enable\_apply\_changes (value)

Sets apply changes flag.

**Parameters value** (Boolean) – *True* to enable apply changes flag, *False* to disable it.

#### enable bluetooth()

Enables the Bluetooth interface of this XBee.

To work with this interface, you must also configure the Bluetooth password if not done previously. Use method  $AbstractXBeeDevice.update\_bluetooth\_password()$ .

Note that your XBee must include Bluetooth Low Energy support.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### execute\_command (parameter, value=None, apply=None)

Executes the provided command.

# **Parameters**

(String or (parameter) - class: .ATStringCommand): AT command to execute.

- value (bytearray, optional, default=`None`) Command value (if any).
- apply (Boolean, optional, default=`None`) True to apply changes in XBee configuration, False not to apply them, None to use is\_apply\_changes\_enabled() returned value.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
AbstractXBeeDevice.get_parameter()
AbstractXBeeDevice.set_parameter()
AbstractXBeeDevice.apply_changes()
AbstractXBeeDevice.write_changes()
AbstractXBeeDevice.is_apply_changes_enabled()
AbstractXBeeDevice.enable_apply_changes()

get_16bit_addr()
Returns the 16-bit address of the XBee.
Return type XBee16BitAddress
See also:

XBee16BitAddress

get_64bit_addr()
Returns the 64-bit address of the XBee.
```

See also:

XBee64BitAddress

## get\_adc\_value(io\_line)

Returns the analog value of the provided IO line.

**Returns** 64-bit address of the XBee. **Return type** XBee64BitAddress

The provided IO line must be previously configured as ADC. To do so, use <code>AbstractXBeeDevice.set\_io\_configuration()</code> and <code>IOMode.ADC</code>.

**Parameters** io\_line (IOLine) – IO line to get its ADC value.

**Returns** Analog value corresponding to the provided IO line.

Return type Integer

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

## See also:

```
IOLine
set_io_configuration()
```

## get\_api\_output\_mode()

Deprecated since version 1.3: Use get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format of the data through the serial interface of the XBee.

**Returns** API output mode of the XBee.

Return type APIOutputMode

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
APIOutputMode
```

## get\_api\_output\_mode\_value()

Returns the API output mode of the XBee.

The API output mode determines the format that the received data is output through the serial interface of the XBee.

**Returns** the parameter value.

**Return type** Bytearray

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

## See also:

```
digi.xbee.models.mode.APIOutputModeBit
```

### get bluetooth mac addr()

Reads and returns the EUI-48 Bluetooth MAC address of this XBee following the format 00112233AABB.

Note that your device must include Bluetooth Low Energy support.

Returns The Bluetooth MAC address.

**Return type** String

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_comm\_iface()

Returns the communication interface of the local XBee associated to the remote one.

## Returns

Communication interface of the local XBee associated to the remote one.

Return type XBeeCommunicationInterface

## See also:

XBeeCommunicationInterface

# get\_current\_frame\_id()

Returns the last used frame ID.

**Returns** Last used frame ID.

Return type Integer

# get\_dest\_address()

Returns the 64-bit address of the XBee that is data destination.

**Returns** 64-bit address of destination XBee.

Return type XBee64BitAddress

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
XBee64BitAddress
set_dest_address()
```

# get\_dio\_value(io\_line)

Returns the digital value of the provided IO line.

The provided IO line must be previously configured as digital I/O. To do so, use AbstractXBeeDevice.set\_io\_configuration().

**Parameters** io\_line (IOLine) – the DIO line to gets its digital value.

**Returns** current value of the provided IO line.

Return type IOValue

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If response does not contain the value for the given IO line.

## See also:

```
IOLine
IOValue
set_io_configuration()

get_file_manager()
```

Returns the file system manager for the XBee.

**Returns** The file system manager.

Return type FileSystemManager

 $\label{lem:continuous} \textbf{Raises} \ \texttt{FileSystemNotSupportedException} - If \ the \ XBee \ does \ not \ support \ filesystem.$ 

# get\_firmware\_version()

Returns the firmware version of the XBee.

**Returns** Firmware version of the XBee.

Return type Bytearray

# get\_hardware\_version()

Returns the hardware version of the XBee.

**Returns** Hardware version of the XBee.

Return type HardwareVersion

See also:

**HardwareVersion** 

# get\_io\_configuration(io\_line)

Returns the configuration of the provided IO line.

**Parameters** io\_line (*IOLine*) – IO line to get its configuration.

Returns IO mode of the IO line provided.

Return type IOMode

**Raises** 

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
IOLine
IOMode
set_io_configuration()
```

## get\_io\_sampling\_rate()

Returns the IO sampling rate of the XBee.

**Returns** IO sampling rate of XBee.

Return type Integer

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

See also:

```
get_local_xbee_device()

Returns the local XBee associated to the remote one.

Returns Local XBee.

Return type XBeeDevice

get_node_id()
```

Returns the node identifier ('NI') value of the XBee.

Returns Node identifier ('NI') of the XBee.

Return type String

```
get_ota_max_block_size()
```

Returns the maximum number of bytes to send for ota updates.

**Returns** Maximum ota block size to send.

Return type Integer

```
get_pan_id()
```

Returns the operating PAN ID of the XBee.

**Returns** Operating PAN ID of the XBee.

Return type Bytearray

**Raises** 

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

See also:

```
set_pan_id()

get_parameter(parameter, parameter_value=None, apply=None)
   Override.
   See also:

AbstractXBeeDevice.get_parameter()
```

```
get_power_level()
```

Returns the power level of the XBee.

**Returns** Power level of the XBee.

Return type PowerLevel

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
PowerLevel
set_power_level()
```

```
get_pwm_duty_cycle(io_line)
```

Returns the PWM duty cycle in % corresponding to the provided IO line.

**Parameters** io\_line (IOLine) - IO line to get its PWM duty cycle.

Returns PWM duty cycle of the given IO line.

Return type Integer

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If io\_line has no PWM capability.

# See also:

*IOLine* 

```
get_role()
```

Gets the XBee role.

**Returns** the role of the XBee.

Return type Role

See also:

Role

```
get_serial_port()
     Returns the serial port of the local XBee associated to the remote one.
          Returns
              Serial port of the local XBee associated to the remote one.
          Return type XBeeSerialPort
     See also:
     XBeeSerialPort
get_sync_ops_timeout()
     Returns the serial port read timeout.
          Returns Serial port read timeout in seconds.
          Return type Integer
is_apply_changes_enabled()
     Returns whether apply changes flag is enabled.
          Returns True if apply changes flag is enabled, False otherwise.
          Return type Boolean
is_remote()
     Override method.
     See also:
     AbstractXBeeDevice.is_remote()
log
     Returns the XBee logger.
          Returns The XBee device logger.
          Return type Logger
reachable
     Returns whether the XBee is reachable.
          Returns True if the device is reachable, False otherwise.
          Return type Boolean
read_device_info (init=True, fire_event=True)
     Updates all instance parameters reading them from the XBee.
          Parameters
                • init (Boolean, optional, default=`True`) - If False only not initial-
                  ized parameters are read, all if True.
```

2.6. API reference 865

and update event if any parameter changed, False otherwise.

Raises

• fire\_event (Boolean, optional, default=`True`) - True to throw

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
AbstractXBeeDevice.is_device_info_complete()
```

## read\_io\_sample()

Returns an IO sample from the XBee containing the value of all enabled digital IO and analog input channels.

**Returns** IO sample read from the XBee.

Return type IOSample

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

```
IOSample
```

## reset()

Override method.

See also:

```
AbstractXBeeDevice.reset()
```

# scan counter

Returns the scan counter for this node.

**Returns** The scan counter for this node.

Return type Integer

## set\_16bit\_addr(value)

Sets the 16-bit address of the XBee.

**Parameters value** (XBee16BitAddress) – New 16-bit address of the XBee.

Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If the protocol is not 802.15.4.

# set\_api\_output\_mode(api\_output\_mode)

Deprecated since version 1.3: Use set\_api\_output\_mode\_value()

Sets the API output mode of the XBee.

**Parameters** api\_output\_mode (APIOutputMode) - New API output mode.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

### See also:

*APIOutputMode* 

# set\_api\_output\_mode\_value(api\_output\_mode)

Sets the API output mode of the XBee.

Parameters api\_output\_mode (Integer) - New API output mode options. Calculate this value using the method APIOutputModeBit. calculate\_api\_output\_mode\_value() with a set of APIOutputModeBit.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- OperationNotSupportedException If it is not supported by the current protocol.

# See also:

*APIOutputModeBit* 

### set dest address(addr)

Sets the 64-bit address of the XBee that is data destination.

**Parameters addr** (XBee64BitAddress or RemoteXBeeDevice) – Address itself or remote XBee to be data destination.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If addr is None.

## See also:

```
XBee64BitAddress
get_dest_address()
```

# set\_dio\_change\_detection(io\_lines\_set)

Sets the digital IO lines to be monitored and sampled whenever their status changes. A *None* set of lines disables this feature.

Parameters io lines set - Set of IOLine.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

*IOLine* 

# set\_dio\_value (io\_line, io\_value)

Sets the digital value (high or low) to the provided IO line.

### **Parameters**

- io\_line (IOLine) Digital IO line to sets its value.
- io\_value (IOValue) IO value to set to the IO line.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

```
IOLine
IOValue
```

# set\_io\_configuration (io\_line, io\_mode)

Sets the configuration of the provided IO line.

## **Parameters**

- io\_line (IOLine) IO line to configure.
- io\_mode (IOMode) IO mode to set to the IO line.

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
IOLine
IOMode
get_io_configuration()
```

# set\_io\_sampling\_rate(rate)

Sets the IO sampling rate of the XBee in seconds. A sample rate of 0 means the IO sampling feature is disabled.

**Parameters** rate (Integer) – New IO sampling rate of the XBee in seconds.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_io_sampling_rate()
```

### set local xbee device (local xbee device)

This methods associates a XBeeDevice to the remote XBee.

**Parameters** local\_xbee\_device (XBeeDevice) - New local XBee associated to the remote one.

## See also:

XBeeDevice

## set\_node\_id (node\_id)

Sets the node identifier ('NI') value of the XBee.

**Parameters node\_id** (String) – New node identifier ('NI') of the XBee.

## **Raises**

- ValueError If *node\_id* is *None* or its length is greater than 20.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# set\_ota\_max\_block\_size(size)

Sets the maximum number of bytes to send for ota updates.

**Parameters** size (Integer) – Maximum ota block size to send.

Raises ValueError – If size is not between 0 and 255.

## set\_pan\_id(value)

Sets the operating PAN ID of the XBee.

**Parameters value** (Bytearray) – New operating PAN ID of the XBee. Must have only 1 or 2 bytes.

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

```
get_pan_id()
```

# set\_parameter (parameter, value, apply=None)

Override.

### See also:

```
AbstractXBeeDevice.set parameter()
```

# set\_power\_level (power\_level)

Sets the power level of the XBee.

**Parameters** power\_level (*PowerLevel*) – New power level of the XBee.

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- $\bullet$  ATCommandException If response is not as expected.

## See also:

```
PowerLevel
get_power_level()
```

# set\_pwm\_duty\_cycle (io\_line, cycle)

Sets the duty cycle in % of the provided IO line.

The provided IO line must be PWM-capable, previously configured as PWM output.

### **Parameters**

- io\_line (IOLine) IO Line to be assigned.
- cycle (Integer) Duty cycle in % to be assigned. Must be between 0 and 100.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.
- ValueError If the given IO line does not have PWM capability or *cycle* is not between 0 and 100.

# See also:

```
IOLine
IOMode.PWM
```

## set\_sync\_ops\_timeout (sync\_ops\_timeout)

Sets the serial port read timeout.

**Parameters** sync\_ops\_timeout (Integer) - Read timeout in seconds.

## update\_bluetooth\_password(new\_password)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

**Parameters** new\_password (String) - New Bluetooth password.

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# update\_device\_data\_from(device)

Updates the current node information with provided data. This is only for internal use.

**Parameters** device (AbstractXBeeDevice) – XBee to get the data from.

**Returns** *True* if the node data has been updated, *False* otherwise.

Return type Boolean

**update\_filesystem\_image** (*ota\_filesystem\_file*, *timeout=None*, *progress\_callback=None*) Performs a filesystem image update operation of the device.

#### **Parameters**

- ota\_filesystem\_file (String) Location of the OTA filesystem image file.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the update process.
- **progress\_callback** (Function, optional) Function to receive progress information. Receives two arguments:
  - The current update task as a String.
  - The current update task percentage as an Integer.

# Raises

- XBeeException If the device is not open.
- InvalidOperatingModeException If the device operating mode is invalid.
- FileSystemNotSupportedException If the filesystem update is not supported in the XBee.
- FileSystemException If there is any error performing the filesystem update.

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_file}, & xbee\_firmware\_file=None, & bootloader\_firmware\_file=None, \\ & timeout=None, & progress\_callback=None) \\ \end{tabular}$ 

Performs a firmware update operation of the XBee.

## **Parameters**

- xml\_firmware\_file (String) Path of the XML file that describes the firmware to upload.
- **xbee\_firmware\_file** (String, optional, default=`None`) Location of the XBee binary firmware file.

- bootloader\_firmware\_file (String, optional, default=`None`) Location of the bootloader binary firmware file.
- timeout (Integer, optional, default=`None`) Maximum time to wait for target read operations during the update process (seconds).
- progress\_callback (Function, optional, default=`None`) Function to to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

## **Raises**

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- OperationNotSupportedException If XBee does not support firmware update.
- FirmwareUpdateException If there is any error during the firmware update.

# write\_changes()

Writes configurable parameter values to the non-volatile memory of the XBee so that parameter modifications persist through subsequent resets.

Parameters values remain in the device's memory until overwritten by subsequent use of this method.

If changes are made without writing them, the XBee reverts back to previously saved parameters the next time the module is powered-on.

Writing the parameter modifications does not mean those values are immediately applied, this depends on the status of the 'apply configuration changes' option. Use method  $is\_apply\_changes\_enabled()$  to get its status and  $enable\_apply\_changes()$  to enable/disable the option. Method  $apply\_changes()$  can be used in order to manually apply the changes.

## **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

class digi.xbee.devices.XBeeNetwork(xbee\_device)

Bases: object

This class represents an XBee Network.

The network allows the discovery of remote devices in the same network as the local one and stores them.

Class constructor. Instantiates a new XBeeNetwork.

**Parameters xbee\_device** (XBeeDevice) – Local XBee to get the network from.

**Raises** ValueError – If *xbee\_device* is *None*.

# ND\_PACKET\_FINISH = 1

Flag that indicates a "discovery process finish" packet.

## ND PACKET REMOTE = 2

Flag that indicates a discovery process packet with info about a remote XBee.

## DEFAULT\_TIME\_BETWEEN\_SCANS = 10

Default time (in seconds) to wait before starting a new scan.

## MIN\_TIME\_BETWEEN\_SCANS = 0

Low limit for the time (in seconds) to wait before starting a new scan.

#### MAX TIME BETWEEN SCANS = 259200

High limit for the time (in seconds) to wait before starting a new scan.

## DEFAULT TIME BETWEEN REQUESTS = 5

Default time (in seconds) to wait between node neighbors requests.

## MIN\_TIME\_BETWEEN\_REQUESTS = 0

Low limit for the time (in seconds) to wait between node neighbors requests.

## MAX\_TIME\_BETWEEN\_REQUESTS = 600

High limit for the time (in seconds) to wait between node neighbors requests.

### SCAN TIL CANCEL = 0

The neighbor discovery process continues until is manually stopped.

### scan\_counter

Returns the scan counter.

**Returns** The scan counter.

Return type Integer

# start\_discovery\_process (deep=False, n\_deep\_scans=1)

Starts the discovery process. This method is not blocking.

This process can discover node neighbors and connections, or only nodes:

• Deep discovery: Network nodes and connections between them (including quality) are discovered.

The discovery process will be running the number of scans configured in *n\_deep\_scans*. A scan is considered the process of discovering the full network. If there are more than one number of scans configured, after finishing one another is started, until *n\_deep\_scans* is satisfied.

See  $set\_deep\_discovery\_options$  () to establish the way the network discovery process is performed.

• No deep discovery: Only network nodes are discovered.

The discovery process will be running until the configured timeout expires or, in case of 802.15.4, until the 'end' packet is read.

It may occur that, after timeout expiration, there are nodes that continue sending discovery responses to the local XBee. In this case, these nodes will not be added to the network.

In 802.15.4, both (deep and no deep discovery) are the same and none discover the node connections or their quality. The difference is the possibility of running more than one scan using a deep discovery.

### **Parameters**

- **deep** (Boolean, optional, default=`False`) True for a deep network scan, looking for neighbors and their connections, False otherwise.
- n\_deep\_scans (Integer, optional, default=1) Number of scans to perform before automatically stopping the discovery process. SCAN\_TIL\_CANCEL means the process will not be automatically stopped. Only applicable if deep=True.

## See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.get_deep_discovery_options()
XBeeNetwork.set_deep_discovery_options()
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.get_discovery_options()
XBeeNetwork.get_discovery_options()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
```

# stop\_discovery\_process()

Stops the discovery process if it is running.

Note that some DigiMesh/DigiPoint devices are blocked until the discovery time configured ('NT' parameter) has elapsed, so, when trying to get/set any parameter during the discovery process, a TimeoutException is raised.

# discover\_device (node\_id)

Blocking method. Discovers and reports the first remote XBee that matches the supplied identifier.

**Parameters node id** (String) – Node identifier of the node to discover.

Returns

**Discovered remote XBee,** *None* if the timeout expires and the node was not found.

```
Return type RemoteXBeeDevice
```

## See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_options()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

## discover\_devices (device\_id\_list)

Blocking method. Attempts to discover a list of nodes and add them to the current network.

This method does not guarantee that all nodes of *device\_id\_list* will be found, even if they exist physically. This depends on the node discovery operation and timeout.

**Parameters** device\_id\_list (List) - List of device IDs to discover.

### Returns

**List with the discovered nodes. It may not contain all nodes** specified in *de-vice\_id\_list*.

Return type List

## See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

# is\_discovery\_running()

Returns whether the discovery process is running.

**Returns** *True* if the discovery process is running, *False* otherwise.

Return type Boolean

# get\_devices()

Returns a copy of the XBee devices list of the network.

If a new XBee node is added to the list after the execution of this method, this new XBee is not added to the list returned by this method.

**Returns** A copy of the XBee devices list of the network.

Return type List

## has devices()

Returns whether there is any device in the network.

#### Returns

True if there is at least one node in the network, False otherwise.

**Return type** Boolean

## get\_number\_devices()

Returns the number of nodes in the network.

**Returns** Number of nodes in the network.

Return type Integer

```
export (dir_path=None, name=None, desc=None)
```

Exports this network to the given file path.

If the provided path already exists the file is removed.

# Params:

**dir\_path** (String, optional, default='None'): Absolute path of the directory to export the network. It should not include the file name. If not defined home directory is used.

name (String, optional, default='None'): Network human readable name. desc (String, optional, default='None'): Network description.

# Returns

**Tuple with result (0: success, 1: failure)** and string (exported file path if success, error string otherwise).

Return type Tuple (Integer, String)

### add network modified callback (callback)

Adds a callback for the event NetworkModified.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The event type as a NetworkEventType.
- The reason of the event as a NetworkEventReason.
- The node added, updated or removed from the network as a XBeeDevice or RemoteXBeeDevice.

## See also:

```
XBeeNetwork.del_network_modified_callback()
```

# add\_device\_discovered\_callback(callback)

Adds a callback for the event DeviceDiscovered.

**Parameters** callback (Function) – The callback. Receives one argument.

• The discovered remote XBee as a Remote XBee Device.

## See also:

```
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

# add\_init\_discovery\_scan\_callback(callback)

Adds a callback for the event InitDiscoveryScan.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan to start (starting with 1).
- Total number of scans.

# See also:

```
XBeeNetwork.del_init_discovery_scan_callback()
```

## add\_end\_discovery\_scan\_callback (callback)

Adds a callback for the event *EndDiscoveryScan*.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan that has finished (starting with 1).
- Total number of scans.

## See also:

```
XBeeNetwork.del_end_discovery_scan_callback()
```

## add\_discovery\_process\_finished\_callback(callback)

Adds a callback for the event DiscoveryProcessFinished.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The event code as an NetworkDiscoveryStatus.
- (Optional) A description of the discovery process as a string.

## See also:

```
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del_device_discovered_callback()
```

# add\_packet\_received\_from\_callback (node, callback)

Adds a callback to listen to any received packet from the provided node.

#### **Parameters**

- node (RemoteXBeeDevice) The node to listen for frames.
- callback (Function) The callback. Receives two arguments.
  - The received packet as a XBeeAPIPacket.
  - The remote XBee who sent the packet as a RemoteXBeeDevice.

### See also:

```
XBeeNetwork.del_packet_received_from_callback()
```

## del network modified callback(callback)

Deletes a callback for the callback list of NetworkModified.

**Parameters** callback (Function) – The callback to delete.

# See also:

```
XBeeNetwork.add_network_modified_callback()
```

## del\_device\_discovered\_callback (callback)

Deletes a callback for the callback list of <code>DeviceDiscovered</code> event.

Parameters callback (Function) - The callback to delete.

## See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

# del\_init\_discovery\_scan\_callback (callback)

Deletes a callback for the callback list of InitDiscoveryScan.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeNetwork.add_init_discovery_scan_callback()
```

# del\_end\_discovery\_scan\_callback(callback)

Deletes a callback for the callback list of *EndDiscoveryScan*.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeNetwork.add_end_discovery_scan_callback()
```

# ${\tt del\_discovery\_process\_finished\_callback}\ (callback)$

Deletes a callback for the callback list of DiscoveryProcessFinished event.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del_device_discovered_callback()
```

# ${\tt del\_packet\_received\_from\_callback} \ (node, callb = None)$

Deletes a received packet callback from the provided node.

# **Parameters**

- node (RemoteXBeeDevice) The node to listen for frames.
- callb (Function, optional, default=`None`) The callback to delete, None to delete all.

See also:

```
XBeeNetwork.add_packet_received_from_callback()
```

clear()

Removes all remote XBee nodes from the network.

### get discovery options()

Returns the network discovery process options.

**Returns** Discovery options value.

Return type Bytearray

## Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# set\_discovery\_options(options)

Configures the discovery options (NO parameter) with the given value.

**Parameters options** (Set of *DiscoveryOptions*) – New discovery options, empty set to clear the options.

#### Raises

- ValueError If options is None.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

DiscoveryOptions

# get\_deep\_discovery\_options()

Returns the deep discovery process options.

### Returns

(NeighborDiscoveryMode, Boolean): Tuple containing:

- mode (NeighborDiscoveryMode): Neighbor discovery mode, the way to perform the network discovery process.
- **remove\_nodes** (**Boolean**): *True* **to remove nodes from the** network if they were not discovered in the last scan, *False* otherwise.

Return type Tuple

# See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

## **Parameters**

- deep\_mode (NeighborDiscoveryMode, optional, default='NeighborDiscoveryMode.CASCADE') Neighbor discovery mode, the way to perform the network discovery process.
- del\_not\_discovered\_nodes\_in\_last\_scan (Boolean, optional, default=`False`) True to remove nodes from the network if they were not discovered in the last scan.

#### See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# get\_discovery\_timeout()

Returns the network discovery timeout.

**Returns** Network discovery timeout.

**Return type** Float

## Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API
  or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# set\_discovery\_timeout (discovery\_timeout)

Sets the discovery network timeout.

**Parameters** discovery\_timeout (Float) - Timeout in seconds.

# Raises

- ValueError If *discovery\_timeout* is not between the allowed minimum and maximum values.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_deep\_discovery\_timeouts()

Gets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see  $start\_discovery\_process()$ )

### Returns

# **Tuple containing:**

- node\_timeout (Float): Maximum duration in seconds of the discovery process per node. This is used to find neighbors of a node. This timeout is highly dependent on the nature of the network:
  - It should be greater than the highest 'NT' (Node Discovery Timeout) of your network.
  - And include enough time to let the message propagate depending on the sleep cycle of your network nodes.
- time\_bw\_nodes (Float): Time to wait between node neighbors requests. Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float): Time to wait before starting a new network scan.

**Return type** Tuple (Float, Float, Float)

## See also:

```
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

```
set_deep_discovery_timeouts(node_timeout=None,
```

time\_bw\_requests=None,

 $time\_bw\_scans=None)$  Sets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see  $start\_discovery\_process()$ )

- **node\_timeout** (**Float, optional, default='None'**): Maximum duration in seconds of the discovery process used to find neighbors of a node. If *None* already configured timeouts are used.
- time\_bw\_requests (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_REQUESTS'): Time to wait between node neighbors requests. It must be between MIN\_TIME\_BETWEEN\_REQUESTS and MAX\_TIME\_BETWEEN\_REQUESTS seconds inclusive. Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_SCANS'): Time to wait before starting a new network scan. It must be between MIN\_TIME\_BETWEEN\_SCANS and MAX\_TIME\_BETWEEN\_SCANS seconds inclusive.

**Raises** ValueError – if *node\_timeout*, *time\_bw\_requests* or *time\_bw\_scans* are not between their corresponding limits.

# See also:

```
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

## classmethod get\_nt\_limits(protocol)

Returns a tuple with the minimum and maximum values for the 'NT' value depending on the protocol.

## Returns

Minimum value in seconds, maximum value in seconds.

**Return type** Tuple (Float, Float)

# $is\_node\_in\_network(node)$

Checks if the provided node is in the network or if it is the local XBee.

**Parameters** node (AbstractXBeeDevice) – The node to check.

**Returns** *True* if the node is in the network, *False* otherwise.

Return type Boolean

Raises ValueError - If node is None.

# get\_device\_by\_64 (x64bit\_addr)

Returns the XBee in the network whose 64-bit address matches the given one.

Parameters x64bit\_addr (XBee64BitAddress) - 64-bit address of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x64bit\_addr is None or unknown.

# get\_device\_by\_16 (x16bit\_addr)

Returns the XBee in the network whose 16-bit address matches the given one.

Parameters **x16bit\_addr** (XBee16BitAddress) - 16-bit address of the node to retrieve

**Returns** XBee in the network or *Non* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x16bit\_addr is None or unknown.

# ${\tt get\_device\_by\_node\_id} \, (node\_id)$

Returns the XBee in the network whose node identifier matches the given one.

**Parameters** node\_id (String) – Node identifier of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If node\_id is None.

# add\_if\_not\_exist (x64bit\_addr=None, x16bit\_addr=None, node\_id=None)

Adds an XBee with the provided information if it does not exist in the current network.

If the XBee already exists, its data is updated with the provided information.

If no valid address is provided (x64bit\_addr, x16bit\_addr), None is returned.

# **Parameters**

x64bit\_addr (XBee64BitAddress, optional, default='None') - 64-bit address.

- x16bit\_addr (XBee16BitAddress, optional, default='None') 16-bit address.
- node\_id(String, optional, default=`None`) Node identifier.

#### Returns

the remote XBee with the updated information. If the XBee was not in the list yet, this method returns the given XBee without changes.

Return type AbstractXBeeDevice

## add\_remote (remote\_xbee)

Adds the provided remote XBee to the network if it is not in yet.

If the XBee is already in the network, its data is updated with the information of the provided XBee that are not *None*.

**Parameters** remote\_xbee (RemoteXBeeDevice) - Remote XBee to add.

#### Returns

**Provided XBee with updated data. If** the XBee was not in the list, it returns it without changes.

Return type RemoteXBeeDevice

# add\_remotes (remote\_xbees)

Adds a list of remote XBee nodes to the network.

If any node in the list is already in the network, its data is updated with the information of the corresponding XBee in the list.

**Parameters** remote\_xbees (List) - List of RemoteXBeeDevice to add.

## remove\_device (remote\_xbee)

Removes the provided remote XBee from the network.

Parameters remote xbee (RemoteXBeeDevice) - Remote XBee to remove.

**Raises** ValueError – If the provided *remote\_xbee* is not in the network.

## get\_discovery\_callbacks()

Returns the API callbacks that are used in the device discovery process.

This callbacks notify the user callbacks for each XBee discovered.

### Returns

**Callback for generic devices discovery** process, callback for discovery specific XBee ops.

**Return type** Tuple (Function, Function)

# get\_connections()

Returns a copy of the XBee network connections.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Returns** A copy of the list of *Connection* for the network.

**Return type** List

See also:

884

```
XBeeNetwork.get_node_connections()
XBeeNetwork.start_discovery_process()
```

### get\_node\_connections (node)

Returns the network connections with one of their ends node.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Parameters** node (AbstractXBeeDevice) – The node to get its connections.

Returns List of Connection with node end.

Return type List

## See also:

```
XBeeNetwork.get_connections()
XBeeNetwork.start_discovery_process()
```

# class digi.xbee.devices.ZigBeeNetwork(device)

Bases: digi.xbee.devices.XBeeNetwork

This class represents a Zigbee network.

The network allows the discovery of remote nodes in the same network as the local one and stores them.

Class constructor. Instantiates a new ZigBeeNetwork.

**Parameters** device (ZigBeeDevice) – Local Zigbee node to get the network from.

```
Raises ValueError – If device is None.
```

```
add device discovered callback(callback)
```

Adds a callback for the event DeviceDiscovered.

**Parameters** callback (Function) – The callback. Receives one argument.

• The discovered remote XBee as a RemoteXBeeDevice.

## See also:

```
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

## add\_discovery\_process\_finished\_callback(callback)

Adds a callback for the event DiscoveryProcessFinished.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The event code as an NetworkDiscoveryStatus.
- (Optional) A description of the discovery process as a string.

See also:

```
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del_device_discovered_callback()
```

# add\_end\_discovery\_scan\_callback(callback)

Adds a callback for the event *EndDiscoveryScan*.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan that has finished (starting with 1).
- Total number of scans.

# See also:

```
XBeeNetwork.del_end_discovery_scan_callback()
```

## add\_if\_not\_exist (x64bit\_addr=None, x16bit\_addr=None, node\_id=None)

Adds an XBee with the provided information if it does not exist in the current network.

If the XBee already exists, its data is updated with the provided information.

If no valid address is provided (x64bit\_addr, x16bit\_addr), None is returned.

#### **Parameters**

- x64bit\_addr (XBee64BitAddress, optional, default='None') 64-bit address.
- **x16bit\_addr** (XBee16BitAddress, optional, default='None') 16-bit address.
- node\_id(String, optional, default=`None`) Node identifier.

### Returns

the remote XBee with the updated information. If the XBee was not in the list yet, this method returns the given XBee without changes.

Return type AbstractXBeeDevice

# add\_init\_discovery\_scan\_callback (callback)

Adds a callback for the event InitDiscoveryScan.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan to start (starting with 1).
- Total number of scans.

# See also:

```
XBeeNetwork.del_init_discovery_scan_callback()
```

## add\_network\_modified\_callback (callback)

Adds a callback for the event NetworkModified.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The event type as a NetworkEventType.
- The reason of the event as a NetworkEventReason.
- The node added, updated or removed from the network as a XBeeDevice or RemoteXBeeDevice.

## See also:

```
XBeeNetwork.del_network_modified_callback()
```

## add\_packet\_received\_from\_callback (node, callback)

Adds a callback to listen to any received packet from the provided node.

### **Parameters**

- **node** (RemoteXBeeDevice) The node to listen for frames.
- callback (Function) The callback. Receives two arguments.
  - The received packet as a XBeeAPIPacket.
  - The remote XBee who sent the packet as a RemoteXBeeDevice.

## See also:

```
XBeeNetwork.del_packet_received_from_callback()
```

# add\_remote (remote\_xbee)

Adds the provided remote XBee to the network if it is not in yet.

If the XBee is already in the network, its data is updated with the information of the provided XBee that are not *None*.

**Parameters** remote\_xbee (RemoteXBeeDevice) - Remote XBee to add.

## Returns

**Provided XBee with updated data. If** the XBee was not in the list, it returns it without changes.

Return type RemoteXBeeDevice

## add remotes(remote xbees)

Adds a list of remote XBee nodes to the network.

If any node in the list is already in the network, its data is updated with the information of the corresponding XBee in the list.

Parameters remote\_xbees (List) - List of RemoteXBeeDevice to add.

### clear()

Removes all remote XBee nodes from the network.

# del\_device\_discovered\_callback(callback)

Deletes a callback for the callback list of DeviceDiscovered event.

**Parameters** callback (Function) – The callback to delete.

### See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

# del\_discovery\_process\_finished\_callback(callback)

Deletes a callback for the callback list of <code>DiscoveryProcessFinished</code> event.

**Parameters** callback (Function) – The callback to delete.

#### See also:

```
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del_device_discovered_callback()
```

## del\_end\_discovery\_scan\_callback (callback)

Deletes a callback for the callback list of EndDiscoveryScan.

Parameters callback (Function) - The callback to delete.

## See also:

```
XBeeNetwork.add_end_discovery_scan_callback()
```

## del init discovery scan callback (callback)

Deletes a callback for the callback list of InitDiscoveryScan.

**Parameters** callback (Function) – The callback to delete.

## See also:

```
XBeeNetwork.add_init_discovery_scan_callback()
```

# del\_network\_modified\_callback(callback)

Deletes a callback for the callback list of NetworkModified.

**Parameters** callback (Function) – The callback to delete.

## See also:

```
XBeeNetwork.add_network_modified_callback()
```

# ${\tt del\_packet\_received\_from\_callback} \ (node, callb = None)$

Deletes a received packet callback from the provided node.

## **Parameters**

• node (RemoteXBeeDevice) – The node to listen for frames.

• callb (Function, optional, default=`None`) - The callback to delete. None to delete all.

### See also:

```
XBeeNetwork.add_packet_received_from_callback()
```

# ${\tt discover\_device} \ (node\_id)$

Blocking method. Discovers and reports the first remote XBee that matches the supplied identifier.

**Parameters node\_id** (*String*) – Node identifier of the node to discover.

#### Returns

Discovered remote XBee, None if the timeout expires and the node was not found.

Return type RemoteXBeeDevice

## See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

# discover\_devices (device\_id\_list)

Blocking method. Attempts to discover a list of nodes and add them to the current network.

This method does not guarantee that all nodes of *device\_id\_list* will be found, even if they exist physically. This depends on the node discovery operation and timeout.

**Parameters** device\_id\_list (List) – List of device IDs to discover.

# Returns

**List with the discovered nodes. It may not contain all nodes** specified in *device\_id\_list*.

**Return type** List

## See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_options()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

# export (dir\_path=None, name=None, desc=None)

Exports this network to the given file path.

If the provided path already exists the file is removed.

## **Params:**

**dir\_path** (**String, optional, default='None'): Absolute path of the** directory to export the network. It should not include the file name. If not defined home directory is used.

name (String, optional, default='None'): Network human readable name. desc (String, optional, default='None'): Network description.

## **Returns**

**Tuple with result (0: success, 1: failure)** and string (exported file path if success, error string otherwise).

Return type Tuple (Integer, String)

## get\_connections()

Returns a copy of the XBee network connections.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Returns** A copy of the list of *Connection* for the network.

Return type List

See also:

```
XBeeNetwork.get_node_connections()
XBeeNetwork.start_discovery_process()
```

# get\_deep\_discovery\_options()

Returns the deep discovery process options.

### Returns

(NeighborDiscoveryMode, Boolean): Tuple containing:

- mode (NeighborDiscoveryMode): Neighbor discovery mode, the way to perform the network discovery process.
- **remove\_nodes** (**Boolean**): *True* **to remove nodes from the** network if they were not discovered in the last scan, *False* otherwise.

Return type Tuple

See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# get\_deep\_discovery\_timeouts()

Gets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see  $start\_discovery\_process()$ )

Returns

# **Tuple containing:**

- node\_timeout (Float): Maximum duration in seconds of the discovery process per node. This is used to find neighbors of a node. This timeout is highly dependent on the nature of the network:
  - It should be greater than the highest 'NT' (Node Discovery Timeout) of your network.
  - And include enough time to let the message propagate depending on the sleep cycle of your network nodes.
- time\_bw\_nodes (Float): Time to wait between node neighbors requests.

  Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float): Time to wait before starting a new network scan.

**Return type** Tuple (Float, Float, Float)

#### See also:

```
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# get\_device\_by\_16 (x16bit\_addr)

Returns the XBee in the network whose 16-bit address matches the given one.

Parameters **x16bit\_addr** (XBee16BitAddress) - 16-bit address of the node to retrieve.

**Returns** XBee in the network or *Non* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x16bit\_addr is None or unknown.

# get\_device\_by\_64 (x64bit\_addr)

Returns the XBee in the network whose 64-bit address matches the given one.

**Parameters x64bit\_addr** (XBee64BitAddress) - 64-bit address of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x64bit addr is None or unknown.

## get\_device\_by\_node\_id (node\_id)

Returns the XBee in the network whose node identifier matches the given one.

**Parameters** node\_id (String) - Node identifier of the node to retrieve.

Returns XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If node\_id is None.

### get devices()

Returns a copy of the XBee devices list of the network.

If a new XBee node is added to the list after the execution of this method, this new XBee is not added to the list returned by this method.

**Returns** A copy of the XBee devices list of the network.

**Return type** List

# get\_discovery\_callbacks()

Returns the API callbacks that are used in the device discovery process.

This callbacks notify the user callbacks for each XBee discovered.

#### Returns

**Callback for generic devices discovery** process, callback for discovery specific XBee ops.

Return type Tuple (Function, Function)

## get\_discovery\_options()

Returns the network discovery process options.

**Returns** Discovery options value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_discovery\_timeout()

Returns the network discovery timeout.

**Returns** Network discovery timeout.

Return type Float

### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_node\_connections (node)

Returns the network connections with one of their ends *node*.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Parameters** node (AbstractXBeeDevice) – The node to get its connections.

Returns List of Connection with node end.

**Return type** List

See also:

```
XBeeNetwork.get_connections()
XBeeNetwork.start_discovery_process()
```

# classmethod get\_nt\_limits(protocol)

Returns a tuple with the minimum and maximum values for the 'NT' value depending on the protocol.

#### Returns

Minimum value in seconds, maximum value in seconds.

**Return type** Tuple (Float, Float)

## get\_number\_devices()

Returns the number of nodes in the network.

**Returns** Number of nodes in the network.

Return type Integer

## has devices()

Returns whether there is any device in the network.

## Returns

*True* if there is at least one node in the network, *False* otherwise.

Return type Boolean

# is\_discovery\_running()

Returns whether the discovery process is running.

**Returns** *True* if the discovery process is running, *False* otherwise.

Return type Boolean

## is\_node\_in\_network (node)

Checks if the provided node is in the network or if it is the local XBee.

**Parameters** node (AbstractXBeeDevice) – The node to check.

Returns True if the node is in the network, False otherwise.

Return type Boolean

**Raises** ValueError – If *node* is *None*.

# remove\_device (remote\_xbee)

Removes the provided remote XBee from the network.

Parameters remote xbee (Remote XBee Device) - Remote XBee to remove.

**Raises** ValueError – If the provided *remote\_xbee* is not in the network.

# scan\_counter

Returns the scan counter.

**Returns** The scan counter.

Return type Integer

set\_deep\_discovery\_options(deep\_mode=<NeighborDiscoveryMode.CASCADE: (0, 'Cas-

cade')>, del\_not\_discovered\_nodes\_in\_last\_scan=False)
Configures the deep discovery options with the given values. These options are only applicable for "deep" discovery (see start\_discovery\_process())

#### **Parameters**

- **deep\_mode** (NeighborDiscoveryMode, optional, default='NeighborDiscoveryMode.CASCADE') Neighbor discovery mode, the way to perform the network discovery process.
- del\_not\_discovered\_nodes\_in\_last\_scan (Boolean, optional, default=`False`) True to remove nodes from the network if they were not discovered in the last scan.

### See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

 $\verb|set_deep_discovery_timeouts| (\textit{node\_timeout=None},$ 

time\_bw\_requests=None,

time\_bw\_scans=None)

Sets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see start\_discovery\_process())

- **node\_timeout** (**Float, optional, default='None'):** Maximum duration in seconds of the discovery process used to find neighbors of a node. If *None* already configured timeouts are used.
- time\_bw\_requests (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_REQUESTS'): Time to wait between node neighbors requests. It must be between MIN\_TIME\_BETWEEN\_REQUESTS and MAX\_TIME\_BETWEEN\_REQUESTS seconds inclusive. Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_SCANS'): Time to wait before starting a new network scan. It must be between MIN\_TIME\_BETWEEN\_SCANS and MAX\_TIME\_BETWEEN\_SCANS seconds inclusive.

**Raises** ValueError – if *node\_timeout*, *time\_bw\_requests* or *time\_bw\_scans* are not between their corresponding limits.

### See also:

```
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

#### set discovery options (options)

Configures the discovery options (NO parameter) with the given value.

**Parameters options** (Set of *DiscoveryOptions*) – New discovery options, empty set to clear the options.

# Raises

- ValueError If options is None.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

DiscoveryOptions

# set\_discovery\_timeout (discovery\_timeout)

Sets the discovery network timeout.

**Parameters** discovery\_timeout (Float) - Timeout in seconds.

# Raises

- ValueError If *discovery\_timeout* is not between the allowed minimum and maximum values.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### start discovery process(deep=False, n deep scans=1)

Starts the discovery process. This method is not blocking.

This process can discover node neighbors and connections, or only nodes:

· Deep discovery: Network nodes and connections between them (including quality) are discovered.

The discovery process will be running the number of scans configured in  $n\_deep\_scans$ . A scan is considered the process of discovering the full network. If there are more than one number of scans configured, after finishing one another is started, until  $n\_deep\_scans$  is satisfied.

See set\_deep\_discovery\_options() to establish the way the network discovery process is performed.

• No deep discovery: Only network nodes are discovered.

The discovery process will be running until the configured timeout expires or, in case of 802.15.4, until the 'end' packet is read.

It may occur that, after timeout expiration, there are nodes that continue sending discovery responses to the local XBee. In this case, these nodes will not be added to the network.

In 802.15.4, both (deep and no deep discovery) are the same and none discover the node connections or their quality. The difference is the possibility of running more than one scan using a deep discovery.

#### **Parameters**

- **deep** (Boolean, optional, default=`False`) True for a deep network scan, looking for neighbors and their connections, False otherwise.
- n\_deep\_scans (Integer, optional, default=1) Number of scans to perform before automatically stopping the discovery process. SCAN\_TIL\_CANCEL means the process will not be automatically stopped. Only applicable if deep=True.

#### See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.get_deep_discovery_options()
XBeeNetwork.set_deep_discovery_options()
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.get_discovery_options()
XBeeNetwork.get_discovery_options()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
```

# stop\_discovery\_process()

Stops the discovery process if it is running.

Note that some DigiMesh/DigiPoint devices are blocked until the discovery time configured ('NT' parameter) has elapsed, so, when trying to get/set any parameter during the discovery process, a TimeoutException is raised.

```
class digi.xbee.devices.Raw802Network(xbee_device)
    Bases: digi.xbee.devices.XBeeNetwork
```

This class represents an 802.15.4 network.

The network allows the discovery of remote nodes in the same network as the local one and stores them.

Class constructor. Instantiates a new *XBeeNetwork*.

Parameters xbee\_device (XBeeDevice) - Local XBee to get the network from.

```
Raises ValueError - If xbee_device is None.

add_device_discovered_callback (callback)

Adds a callback for the event DeviceDiscovered.
```

**Parameters** callback (Function) – The callback. Receives one argument.

• The discovered remote XBee as a RemoteXBeeDevice.

#### See also:

```
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

# add\_discovery\_process\_finished\_callback(callback)

Adds a callback for the event DiscoveryProcessFinished.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The event code as an NetworkDiscoveryStatus.
- (Optional) A description of the discovery process as a string.

#### See also:

```
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del device discovered callback()
```

# add\_end\_discovery\_scan\_callback(callback)

Adds a callback for the event EndDiscoveryScan.

**Parameters callback** (Function) – The callback. Receives two arguments.

- Number of scan that has finished (starting with 1).
- Total number of scans.

#### See also:

```
XBeeNetwork.del_end_discovery_scan_callback()
```

# add\_if\_not\_exist (x64bit\_addr=None, x16bit\_addr=None, node\_id=None)

Adds an XBee with the provided information if it does not exist in the current network.

If the XBee already exists, its data is updated with the provided information.

If no valid address is provided (x64bit\_addr, x16bit\_addr), None is returned.

# **Parameters**

- x64bit\_addr (XBee64BitAddress, optional, default='None') 64-bit address
- x16bit\_addr (XBee16BitAddress, optional, default='None') 16-bit address.
- node id(String, optional, default=`None`) Node identifier.

## Returns

**the remote XBee with the updated** information. If the XBee was not in the list yet, this method returns the given XBee without changes.

Return type AbstractXBeeDevice

# add\_init\_discovery\_scan\_callback(callback)

Adds a callback for the event InitDiscoveryScan.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan to start (starting with 1).
- Total number of scans.

# See also:

```
XBeeNetwork.del_init_discovery_scan_callback()
```

#### add network modified callback(callback)

Adds a callback for the event NetworkModified.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The event type as a NetworkEventType.
- The reason of the event as a NetworkEventReason.
- The node added, updated or removed from the network as a XBeeDevice or RemoteXBeeDevice.

#### See also:

```
XBeeNetwork.del_network_modified_callback()
```

# add\_packet\_received\_from\_callback (node, callback)

Adds a callback to listen to any received packet from the provided node.

# **Parameters**

- **node** (RemoteXBeeDevice) The node to listen for frames.
- callback (Function) The callback. Receives two arguments.
  - The received packet as a XBeeAPIPacket.
  - The remote XBee who sent the packet as a RemoteXBeeDevice.

#### See also:

```
XBeeNetwork.del_packet_received_from_callback()
```

## add\_remote (remote\_xbee)

Adds the provided remote XBee to the network if it is not in yet.

If the XBee is already in the network, its data is updated with the information of the provided XBee that are not *None*.

**Parameters** remote\_xbee (RemoteXBeeDevice) - Remote XBee to add.

#### Returns

**Provided XBee with updated data. If** the XBee was not in the list, it returns it without changes.

Return type RemoteXBeeDevice

#### add remotes (remote xbees)

Adds a list of remote XBee nodes to the network.

If any node in the list is already in the network, its data is updated with the information of the corresponding XBee in the list.

**Parameters** remote\_xbees (List) - List of RemoteXBeeDevice to add.

#### clear()

Removes all remote XBee nodes from the network.

# del\_device\_discovered\_callback(callback)

Deletes a callback for the callback list of DeviceDiscovered event.

**Parameters** callback (Function) – The callback to delete.

#### See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

# del\_discovery\_process\_finished\_callback(callback)

Deletes a callback for the callback list of DiscoveryProcessFinished event.

**Parameters** callback (Function) – The callback to delete.

#### See also:

```
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del_device_discovered_callback()
```

# del\_end\_discovery\_scan\_callback(callback)

Deletes a callback for the callback list of EndDiscoveryScan.

Parameters callback (Function) - The callback to delete.

# See also:

```
XBeeNetwork.add_end_discovery_scan_callback()
```

## del init discovery scan callback (callback)

Deletes a callback for the callback list of InitDiscoveryScan.

**Parameters** callback (Function) – The callback to delete.

#### See also:

```
XBeeNetwork.add_init_discovery_scan_callback()
```

#### del network modified callback (callback)

Deletes a callback for the callback list of NetworkModified.

**Parameters** callback (Function) – The callback to delete.

#### See also:

```
XBeeNetwork.add_network_modified_callback()
```

# del\_packet\_received\_from\_callback (node, callb=None)

Deletes a received packet callback from the provided node.

#### **Parameters**

- node (RemoteXBeeDevice) The node to listen for frames.
- callb (Function, optional, default=`None`) The callback to delete, None to delete all.

#### See also:

```
XBeeNetwork.add_packet_received_from_callback()
```

# discover\_device (node\_id)

Blocking method. Discovers and reports the first remote XBee that matches the supplied identifier.

**Parameters** node\_id (String) - Node identifier of the node to discover.

# Returns

Discovered remote XBee, None if the timeout expires and the node was not found.

Return type RemoteXBeeDevice

## See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_options()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

# discover\_devices (device\_id\_list)

Blocking method. Attempts to discover a list of nodes and add them to the current network.

This method does not guarantee that all nodes of *device\_id\_list* will be found, even if they exist physically. This depends on the node discovery operation and timeout.

**Parameters** device\_id\_list (List) – List of device IDs to discover.

#### Returns

**List with the discovered nodes. It may not contain all nodes** specified in *device\_id\_list*.

# Return type List

#### See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

# export (dir\_path=None, name=None, desc=None)

Exports this network to the given file path.

If the provided path already exists the file is removed.

#### **Params:**

**dir\_path** (**String, optional, default='None'): Absolute path of the** directory to export the network. It should not include the file name. If not defined home directory is used.

name (String, optional, default='None'): Network human readable name. desc (String, optional, default='None'): Network description.

#### Returns

**Tuple with result (0: success, 1: failure)** and string (exported file path if success, error string otherwise).

Return type Tuple (Integer, String)

## get\_connections()

Returns a copy of the XBee network connections.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Returns** A copy of the list of *Connection* for the network.

**Return type** List

## See also:

```
XBeeNetwork.get_node_connections()
XBeeNetwork.start_discovery_process()
```

# get\_deep\_discovery\_options()

Returns the deep discovery process options.

## Returns

(NeighborDiscoveryMode, Boolean): Tuple containing:

- mode (NeighborDiscoveryMode): Neighbor discovery mode, the way to perform the network discovery process.
- **remove\_nodes (Boolean):** *True* **to remove nodes from the** network if they were not discovered in the last scan, *False* otherwise.

# Return type Tuple

#### See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# get\_deep\_discovery\_timeouts()

Gets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see start\_discovery\_process())

# Returns

## **Tuple containing:**

- node\_timeout (Float): Maximum duration in seconds of the discovery process per node. This is used to find neighbors of a node. This timeout is highly dependent on the nature of the network:
  - It should be greater than the highest 'NT' (Node Discovery Timeout) of your network.
  - And include enough time to let the message propagate depending on the sleep cycle of your network nodes.
- $\bullet \ time\_bw\_nodes \ (Float) \hbox{:} \ Time \ to \ wait \ between \ node \ neighbors \ \ requests.$

Use this setting not to saturate your network:

- For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
- For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float): Time to wait before starting a new network scan.

Return type Tuple (Float, Float, Float)

#### See also:

```
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# get\_device\_by\_16 (x16bit\_addr)

Returns the XBee in the network whose 16-bit address matches the given one.

Parameters **x16bit\_addr** (XBee16BitAddress) - 16-bit address of the node to retrieve.

**Returns** XBee in the network or *Non* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x16bit\_addr is None or unknown.

# get\_device\_by\_64 (x64bit\_addr)

Returns the XBee in the network whose 64-bit address matches the given one.

Parameters x64bit\_addr (XBee64BitAddress) - 64-bit address of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x64bit\_addr is None or unknown.

#### get\_device\_by\_node\_id (node\_id)

Returns the XBee in the network whose node identifier matches the given one.

**Parameters** node\_id (String) – Node identifier of the node to retrieve.

Returns XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If node\_id is None.

# get\_devices()

Returns a copy of the XBee devices list of the network.

If a new XBee node is added to the list after the execution of this method, this new XBee is not added to the list returned by this method.

**Returns** A copy of the XBee devices list of the network.

**Return type** List

# get\_discovery\_callbacks()

Returns the API callbacks that are used in the device discovery process.

This callbacks notify the user callbacks for each XBee discovered.

#### **Returns**

**Callback for generic devices discovery** process, callback for discovery specific XBee ops.

Return type Tuple (Function, Function)

# get\_discovery\_options()

Returns the network discovery process options.

**Returns** Discovery options value.

**Return type** Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### get\_discovery\_timeout()

Returns the network discovery timeout.

**Returns** Network discovery timeout.

Return type Float

#### **Raises**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_node\_connections (node)

Returns the network connections with one of their ends *node*.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Parameters** node (AbstractXBeeDevice) – The node to get its connections.

Returns List of Connection with node end.

Return type List

See also:

```
XBeeNetwork.get_connections()
XBeeNetwork.start_discovery_process()
```

# classmethod get\_nt\_limits(protocol)

Returns a tuple with the minimum and maximum values for the 'NT' value depending on the protocol.

#### Returns

Minimum value in seconds, maximum value in seconds.

**Return type** Tuple (Float, Float)

# get\_number\_devices()

Returns the number of nodes in the network.

**Returns** Number of nodes in the network.

Return type Integer

# has\_devices()

Returns whether there is any device in the network.

#### **Returns**

True if there is at least one node in the network, False otherwise.

**Return type** Boolean

#### is\_discovery\_running()

Returns whether the discovery process is running.

**Returns** *True* if the discovery process is running, *False* otherwise.

Return type Boolean

#### is\_node\_in\_network (node)

Checks if the provided node is in the network or if it is the local XBee.

**Parameters node** (AbstractXBeeDevice) – The node to check.

**Returns** *True* if the node is in the network, *False* otherwise.

Return type Boolean

**Raises** ValueError – If node is None.

# remove\_device (remote\_xbee)

Removes the provided remote XBee from the network.

Parameters remote\_xbee (RemoteXBeeDevice) - Remote XBee to remove.

**Raises** ValueError – If the provided *remote\_xbee* is not in the network.

#### scan counter

Returns the scan counter.

Returns The scan counter.

Return type Integer

Configures the deep discovery options with the given values. These options are only applicable for "deep" discovery (see start\_discovery\_process())

## **Parameters**

- deep\_mode (NeighborDiscoveryMode, optional, default='NeighborDiscoveryMode.CASCADE') Neighbor discovery mode, the way to perform the network discovery process.
- del\_not\_discovered\_nodes\_in\_last\_scan (Boolean, optional, default=`False`) True to remove nodes from the network if they were not discovered in the last scan.

#### See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

Sets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see start discovery process())

**node\_timeout** (**Float, optional, default='None'):** Maximum duration in seconds of the discovery process used to find neighbors of a node. If *None* already configured timeouts are used.

- time\_bw\_requests (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_REQUESTS'): Time to wait between node neighbors requests. It must be between MIN\_TIME\_BETWEEN\_REQUESTS and MAX\_TIME\_BETWEEN\_REQUESTS seconds inclusive. Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.

# time\_bw\_scans (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_SCANS'): Time to wait before starting a new network scan. It must be between MIN\_TIME\_BETWEEN\_SCANS and MAX\_TIME\_BETWEEN\_SCANS seconds inclusive.

**Raises** ValueError – if *node\_timeout*, *time\_bw\_requests* or *time\_bw\_scans* are not between their corresponding limits.

#### See also:

```
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# set\_discovery\_options(options)

Configures the discovery options (NO parameter) with the given value.

**Parameters options** (Set of *DiscoveryOptions*) – New discovery options, empty set to clear the options.

#### Raises

- ValueError If options is None.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## See also:

DiscoveryOptions

# set\_discovery\_timeout (discovery\_timeout)

Sets the discovery network timeout.

**Parameters** discovery\_timeout (Float) - Timeout in seconds.

#### Raises

• ValueError – If *discovery\_timeout* is not between the allowed minimum and maximum values.

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

```
start_discovery_process (deep=False, n_deep_scans=1)
```

Starts the discovery process. This method is not blocking.

This process can discover node neighbors and connections, or only nodes:

• Deep discovery: Network nodes and connections between them (including quality) are discovered.

The discovery process will be running the number of scans configured in  $n\_deep\_scans$ . A scan is considered the process of discovering the full network. If there are more than one number of scans configured, after finishing one another is started, until  $n\_deep\_scans$  is satisfied.

See set\_deep\_discovery\_options() to establish the way the network discovery process is performed.

• No deep discovery: Only network nodes are discovered.

The discovery process will be running until the configured timeout expires or, in case of 802.15.4, until the 'end' packet is read.

It may occur that, after timeout expiration, there are nodes that continue sending discovery responses to the local XBee. In this case, these nodes will not be added to the network.

In 802.15.4, both (deep and no deep discovery) are the same and none discover the node connections or their quality. The difference is the possibility of running more than one scan using a deep discovery.

#### **Parameters**

- **deep** (Boolean, optional, default=`False`) True for a deep network scan, looking for neighbors and their connections, False otherwise.
- n\_deep\_scans (Integer, optional, default=1) Number of scans to perform before automatically stopping the discovery process. SCAN\_TIL\_CANCEL means the process will not be automatically stopped. Only applicable if deep=True.

#### See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.get_deep_discovery_options()
XBeeNetwork.set_deep_discovery_options()
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_options()
XBeeNetwork.set_discovery_timeout()
```

```
XBeeNetwork.set discovery timeout()
```

# stop\_discovery\_process()

Stops the discovery process if it is running.

Note that some DigiMesh/DigiPoint devices are blocked until the discovery time configured ('NT' parameter) has elapsed, so, when trying to get/set any parameter during the discovery process, a TimeoutException is raised.

```
class digi.xbee.devices.DigiMeshNetwork(device)
```

```
Bases: digi.xbee.devices.XBeeNetwork
```

This class represents a DigiMesh network.

The network allows the discovery of remote nodes in the same network as the local one and stores them.

Class constructor. Instantiates a new *DigiMeshNetwork*.

Parameters device (DigiMeshDevice) - Local DigiMesh node to get the network from.

**Raises** ValueError – If device is None.

# add\_device\_discovered\_callback(callback)

Adds a callback for the event DeviceDiscovered.

**Parameters** callback (Function) – The callback. Receives one argument.

• The discovered remote XBee as a RemoteXBeeDevice.

#### See also:

```
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

#### add discovery process finished callback (callback)

Adds a callback for the event DiscoveryProcessFinished.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The event code as an NetworkDiscoveryStatus.
- (Optional) A description of the discovery process as a string.

#### See also:

```
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del_device_discovered_callback()
```

# $\verb"add_end_discovery_scan_callback" (callback)$

Adds a callback for the event EndDiscoveryScan.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan that has finished (starting with 1).
- Total number of scans.

#### See also:

```
XBeeNetwork.del_end_discovery_scan_callback()
```

# add\_if\_not\_exist (x64bit\_addr=None, x16bit\_addr=None, node\_id=None)

Adds an XBee with the provided information if it does not exist in the current network.

If the XBee already exists, its data is updated with the provided information.

If no valid address is provided (x64bit\_addr, x16bit\_addr), None is returned.

#### **Parameters**

- x64bit\_addr (XBee64BitAddress, optional, default='None') 64-bit address.
- x16bit\_addr (XBee16BitAddress, optional, default='None') 16-bit address.
- node\_id(String, optional, default=`None`) Node identifier.

#### Returns

the remote XBee with the updated information. If the XBee was not in the list yet, this method returns the given XBee without changes.

Return type AbstractXBeeDevice

# add\_init\_discovery\_scan\_callback(callback)

Adds a callback for the event InitDiscoveryScan.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan to start (starting with 1).
- Total number of scans.

## See also:

```
XBeeNetwork.del_init_discovery_scan_callback()
```

#### add network modified callback(callback)

Adds a callback for the event NetworkModified.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The event type as a NetworkEventType.
- The reason of the event as a NetworkEventReason.
- The node added, updated or removed from the network as a XBeeDevice or RemoteXBeeDevice.

#### See also:

```
XBeeNetwork.del_network_modified_callback()
```

#### add\_packet\_received\_from\_callback (node, callback)

Adds a callback to listen to any received packet from the provided node.

#### **Parameters**

- node (RemoteXBeeDevice) The node to listen for frames.
- callback (Function) The callback. Receives two arguments.
  - The received packet as a XBeeAPIPacket.
  - The remote XBee who sent the packet as a RemoteXBeeDevice.

#### See also:

```
XBeeNetwork.del_packet_received_from_callback()
```

## add\_remote (remote\_xbee)

Adds the provided remote XBee to the network if it is not in yet.

If the XBee is already in the network, its data is updated with the information of the provided XBee that are not *None*.

Parameters remote\_xbee (RemoteXBeeDevice) - Remote XBee to add.

#### **Returns**

**Provided XBee with updated data. If** the XBee was not in the list, it returns it without changes.

Return type RemoteXBeeDevice

# add\_remotes (remote\_xbees)

Adds a list of remote XBee nodes to the network.

If any node in the list is already in the network, its data is updated with the information of the corresponding XBee in the list.

Parameters remote\_xbees (List) - List of RemoteXBeeDevice to add.

# clear()

Removes all remote XBee nodes from the network.

# del\_device\_discovered\_callback(callback)

Deletes a callback for the callback list of DeviceDiscovered event.

**Parameters** callback (Function) – The callback to delete.

## See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

# ${\tt del\_discovery\_process\_finished\_callback}\ (\textit{callback})$

Deletes a callback for the callback list of DiscoveryProcessFinished event.

Parameters callback (Function) - The callback to delete.

#### See also:

```
XBeeNetwork.add_device_discovered_callback()
    XBeeNetwork.del device discovered callback()
del_end_discovery_scan_callback(callback)
     Deletes a callback for the callback list of EndDiscoveryScan.
          Parameters callback (Function) – The callback to delete.
    See also:
     XBeeNetwork.add_end_discovery_scan_callback()
del_init_discovery_scan_callback(callback)
    Deletes a callback for the callback list of InitDiscoveryScan.
          Parameters callback (Function) – The callback to delete.
    See also:
    XBeeNetwork.add_init_discovery_scan_callback()
del_network_modified_callback(callback)
    Deletes a callback for the callback list of NetworkModified.
          Parameters callback (Function) - The callback to delete.
    See also:
    XBeeNetwork.add network modified callback()
del_packet_received_from_callback (node, callb=None)
    Deletes a received packet callback from the provided node.
          Parameters
                • node (RemoteXBeeDevice) – The node to listen for frames.
                • callb (Function, optional, default=`None`) - The callback to
                  delete, None to delete all.
     See also:
     XBeeNetwork.add_packet_received_from_callback()
discover_device (node_id)
```

XBeeNetwork.add\_discovery\_process\_finished\_callback()

2.6. API reference 911

**Parameters** node\_id (String) – Node identifier of the node to discover.

Returns

Blocking method. Discovers and reports the first remote XBee that matches the supplied identifier.

**Discovered remote XBee,** *None* if the timeout expires and the node was not found.

```
Return type RemoteXBeeDevice
```

## See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_options()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

#### discover\_devices (device\_id\_list)

Blocking method. Attempts to discover a list of nodes and add them to the current network.

This method does not guarantee that all nodes of *device\_id\_list* will be found, even if they exist physically. This depends on the node discovery operation and timeout.

**Parameters** device id list (List) – List of device IDs to discover.

#### **Returns**

**List with the discovered nodes. It may not contain all nodes** specified in *device id list*.

# Return type List

#### See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

export (dir\_path=None, name=None, desc=None)

Exports this network to the given file path.

If the provided path already exists the file is removed.

# Params:

dir\_path (String, optional, default='None'): Absolute path of the directory to export the network. It should not include the file name. If not defined home directory is used.

name (String, optional, default='None'): Network human readable name. desc (String, optional, default='None'): Network description.

# Returns

**Tuple with result (0: success, 1: failure)** and string (exported file path if success, error string otherwise).

Return type Tuple (Integer, String)

# get\_connections()

Returns a copy of the XBee network connections.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Returns** A copy of the list of *Connection* for the network.

**Return type** List

See also:

```
XBeeNetwork.get_node_connections()
XBeeNetwork.start_discovery_process()
```

# get\_deep\_discovery\_options()

Returns the deep discovery process options.

#### Returns

(NeighborDiscoveryMode, Boolean): Tuple containing:

- mode (NeighborDiscoveryMode): Neighbor discovery mode, the way to perform the network discovery process.
- **remove\_nodes (Boolean):** *True* **to remove nodes from the** network if they were not discovered in the last scan, *False* otherwise.

Return type Tuple

See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# get\_deep\_discovery\_timeouts()

Gets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see  $start\_discovery\_process()$ )

#### **Returns**

# **Tuple containing:**

- node\_timeout (Float): Maximum duration in seconds of the discovery process per node. This is used to find neighbors of a node. This timeout is highly dependent on the nature of the network:
  - It should be greater than the highest 'NT' (Node Discovery Timeout) of your network.
  - And include enough time to let the message propagate depending on the sleep cycle of your network nodes.
- time\_bw\_nodes (Float): Time to wait between node neighbors requests.

  Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.

- For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float): Time to wait before starting a new network scan.

**Return type** Tuple (Float, Float, Float)

#### See also:

```
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# get\_device\_by\_16 (x16bit\_addr)

Returns the XBee in the network whose 16-bit address matches the given one.

**Parameters x16bit\_addr** (XBee16BitAddress) - 16-bit address of the node to retrieve.

**Returns** XBee in the network or *Non* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x16bit\_addr is None or unknown.

# get\_device\_by\_64 (x64bit\_addr)

Returns the XBee in the network whose 64-bit address matches the given one.

Parameters x64bit\_addr (XBee64BitAddress) - 64-bit address of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x64bit\_addr is None or unknown.

# get\_device\_by\_node\_id (node\_id)

Returns the XBee in the network whose node identifier matches the given one.

**Parameters** node\_id (String) – Node identifier of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If node\_id is None.

# get\_devices()

Returns a copy of the XBee devices list of the network.

If a new XBee node is added to the list after the execution of this method, this new XBee is not added to the list returned by this method.

**Returns** A copy of the XBee devices list of the network.

Return type List

# get\_discovery\_callbacks()

Returns the API callbacks that are used in the device discovery process.

This callbacks notify the user callbacks for each XBee discovered.

Returns

**Callback for generic devices discovery** process, callback for discovery specific XBee ops.

Return type Tuple (Function, Function)

# get\_discovery\_options()

Returns the network discovery process options.

**Returns** Discovery options value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_discovery\_timeout()

Returns the network discovery timeout.

**Returns** Network discovery timeout.

Return type Float

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get node connections (node)

Returns the network connections with one of their ends node.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

Parameters node (AbstractXBeeDevice) - The node to get its connections.

**Returns** List of Connection with node end.

Return type List

See also:

```
XBeeNetwork.get_connections()
XBeeNetwork.start_discovery_process()
```

#### classmethod get\_nt\_limits(protocol)

Returns a tuple with the minimum and maximum values for the 'NT' value depending on the protocol.

#### Returns

Minimum value in seconds, maximum value in seconds.

**Return type** Tuple (Float, Float)

#### get\_number\_devices()

Returns the number of nodes in the network.

**Returns** Number of nodes in the network.

**Return type** Integer

## has\_devices()

Returns whether there is any device in the network.

#### **Returns**

True if there is at least one node in the network, False otherwise.

Return type Boolean

# is\_discovery\_running()

Returns whether the discovery process is running.

**Returns** *True* if the discovery process is running, *False* otherwise.

Return type Boolean

# is\_node\_in\_network (node)

Checks if the provided node is in the network or if it is the local XBee.

**Parameters node** (AbstractXBeeDevice) – The node to check.

**Returns** *True* if the node is in the network, *False* otherwise.

Return type Boolean

Raises ValueError - If node is None.

#### remove\_device (remote\_xbee)

Removes the provided remote XBee from the network.

**Parameters** remote\_xbee (RemoteXBeeDevice) - Remote XBee to remove.

**Raises** ValueError – If the provided *remote\_xbee* is not in the network.

#### scan counter

Returns the scan counter.

Returns The scan counter.

Return type Integer

# 

Configures the deep discovery options with the given values. These options are only applicable for "deep" discovery (see start\_discovery\_process())

## **Parameters**

deep\_mode (NeighborDiscoveryMode, optional, default='NeighborDiscoveryMode.CASCADE') – Neighbor discovery mode, the way to perform the network discovery process.

• del\_not\_discovered\_nodes\_in\_last\_scan (Boolean, optional, default=`False`) - True to remove nodes from the network if they were not discovered in the last scan.

#### See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# set\_deep\_discovery\_timeouts (node\_timeout=None,

time\_bw\_requests=None,

time\_bw\_scans=None)

Sets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see start\_discovery\_process())

**node\_timeout** (**Float, optional, default='None'):** Maximum duration in seconds of the discovery process used to find neighbors of a node. If *None* already configured timeouts are used.

- time\_bw\_requests (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_REQUESTS'): Time to wait between node neighbors requests. It must be between MIN\_TIME\_BETWEEN\_REQUESTS and MAX\_TIME\_BETWEEN\_REQUESTS seconds inclusive. Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_SCANS'): Time to wait before starting a new network scan. It must be between MIN\_TIME\_BETWEEN\_SCANS and MAX\_TIME\_BETWEEN\_SCANS seconds inclusive.

**Raises** ValueError – if *node\_timeout*, *time\_bw\_requests* or *time\_bw\_scans* are not between their corresponding limits.

#### See also:

```
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

# set\_discovery\_options(options)

Configures the discovery options (NO parameter) with the given value.

**Parameters options** (Set of *DiscoveryOptions*) – New discovery options, empty set to clear the options.

#### Raises

- ValueError If options is None.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# See also:

DiscoveryOptions

# set\_discovery\_timeout (discovery\_timeout)

Sets the discovery network timeout.

**Parameters** discovery\_timeout (Float) - Timeout in seconds.

#### Raises

- ValueError If discovery\_timeout is not between the allowed minimum and maximum values.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# start\_discovery\_process (deep=False, n\_deep\_scans=1)

Starts the discovery process. This method is not blocking.

This process can discover node neighbors and connections, or only nodes:

• Deep discovery: Network nodes and connections between them (including quality) are discovered.

The discovery process will be running the number of scans configured in *n\_deep\_scans*. A scan is considered the process of discovering the full network. If there are more than one number of scans configured, after finishing one another is started, until *n\_deep\_scans* is satisfied.

See  $set\_deep\_discovery\_options()$  to establish the way the network discovery process is performed.

• No deep discovery: Only network nodes are discovered.

The discovery process will be running until the configured timeout expires or, in case of 802.15.4, until the 'end' packet is read.

It may occur that, after timeout expiration, there are nodes that continue sending discovery responses to the local XBee. In this case, these nodes will not be added to the network.

In 802.15.4, both (deep and no deep discovery) are the same and none discover the node connections or their quality. The difference is the possibility of running more than one scan using a deep discovery.

# **Parameters**

• **deep** (Boolean, optional, default=`False`) – *True* for a deep network scan, looking for neighbors and their connections, *False* otherwise.

• n\_deep\_scans (Integer, optional, default=1) — Number of scans to perform before automatically stopping the discovery process. SCAN\_TIL\_CANCEL means the process will not be automatically stopped. Only applicable if deep=True.

## See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.get_deep_discovery_options()
XBeeNetwork.set_deep_discovery_options()
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.get_discovery_options()
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

## stop\_discovery\_process()

Stops the discovery process if it is running.

Note that some DigiMesh/DigiPoint devices are blocked until the discovery time configured ('NT' parameter) has elapsed, so, when trying to get/set any parameter during the discovery process, a TimeoutException is raised.

```
class digi.xbee.devices.DigiPointNetwork(xbee_device)
    Bases: digi.xbee.devices.XBeeNetwork
```

This class represents a DigiPoint network.

The network allows the discovery of remote nodes in the same network as the local one and stores them.

Class constructor. Instantiates a new XBeeNetwork.

**Parameters xbee\_device** (XBeeDevice) – Local XBee to get the network from.

```
Raises ValueError - If xbee_device is None.

add_device_discovered_callback (callback)

Adds a callback for the event DeviceDiscovered.
```

ids a camback for the event DeviceDiscovered.

**Parameters callback** (Function) – The callback. Receives one argument.

• The discovered remote XBee as a RemoteXBeeDevice.

# See also:

```
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_discovery_process_finished_callback()
```

# add\_discovery\_process\_finished\_callback(callback)

Adds a callback for the event DiscoveryProcessFinished.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The event code as an NetworkDiscoveryStatus.
- (Optional) A description of the discovery process as a string.

## See also:

```
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.del_device_discovered_callback()
```

## add\_end\_discovery\_scan\_callback (callback)

Adds a callback for the event EndDiscoveryScan.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan that has finished (starting with 1).
- Total number of scans.

#### See also:

```
XBeeNetwork.del_end_discovery_scan_callback()
```

# add\_if\_not\_exist(x64bit\_addr=None, x16bit\_addr=None, node\_id=None)

Adds an XBee with the provided information if it does not exist in the current network.

If the XBee already exists, its data is updated with the provided information.

If no valid address is provided (x64bit\_addr, x16bit\_addr), None is returned.

#### **Parameters**

- x64bit\_addr (XBee64BitAddress, optional, default='None') 64-bit address.
- **x16bit\_addr** (XBee16BitAddress, optional, default='None') 16-bit address.
- node\_id(String, optional, default=`None`) Node identifier.

#### **Returns**

**the remote XBee with the updated** information. If the XBee was not in the list yet, this method returns the given XBee without changes.

Return type AbstractXBeeDevice

# add\_init\_discovery\_scan\_callback(callback)

Adds a callback for the event InitDiscoveryScan.

**Parameters** callback (Function) – The callback. Receives two arguments.

- Number of scan to start (starting with 1).
- Total number of scans.

## See also:

```
XBeeNetwork.del_init_discovery_scan_callback()
```

# add\_network\_modified\_callback(callback)

Adds a callback for the event NetworkModified.

**Parameters** callback (Function) – The callback. Receives three arguments.

- The event type as a NetworkEventType.
- The reason of the event as a NetworkEventReason.
- The node added, updated or removed from the network as a XBeeDevice or RemoteXBeeDevice.

#### See also:

```
XBeeNetwork.del_network_modified_callback()
```

#### add\_packet\_received\_from\_callback (node, callback)

Adds a callback to listen to any received packet from the provided node.

#### **Parameters**

- node (RemoteXBeeDevice) The node to listen for frames.
- callback (Function) The callback. Receives two arguments.
  - The received packet as a XBeeAPIPacket.
  - The remote XBee who sent the packet as a RemoteXBeeDevice.

## See also:

```
XBeeNetwork.del_packet_received_from_callback()
```

# add\_remote (remote\_xbee)

Adds the provided remote XBee to the network if it is not in yet.

If the XBee is already in the network, its data is updated with the information of the provided XBee that are not *None*.

Parameters remote xbee (RemoteXBeeDevice) - Remote XBee to add.

## Returns

**Provided XBee with updated data. If** the XBee was not in the list, it returns it without changes.

Return type RemoteXBeeDevice

#### add\_remotes (remote\_xbees)

Adds a list of remote XBee nodes to the network.

If any node in the list is already in the network, its data is updated with the information of the corresponding XBee in the list.

Parameters remote xbees (List) - List of RemoteXBeeDevice to add.

```
clear()
     Removes all remote XBee nodes from the network.
del_device_discovered_callback(callback)
     Deletes a callback for the callback list of <code>DeviceDiscovered</code> event.
          Parameters callback (Function) – The callback to delete.
     See also:
     XBeeNetwork.add_device_discovered_callback()
     XBeeNetwork.add discovery process finished callback()
     XBeeNetwork.del_discovery_process_finished_callback()
del discovery process finished callback (callback)
     Deletes\ a\ callback\ for\ the\ callback\ list\ of\ {\it DiscoveryProcessFinished}\ event.
          Parameters callback (Function) – The callback to delete.
     See also:
     XBeeNetwork.add_discovery_process_finished_callback()
     XBeeNetwork.add device discovered callback()
     XBeeNetwork.del_device_discovered_callback()
del_end_discovery_scan_callback (callback)
     Deletes a callback for the callback list of EndDiscoveryScan.
          Parameters callback (Function) – The callback to delete.
     See also:
     XBeeNetwork.add_end_discovery_scan_callback()
del_init_discovery_scan_callback (callback)
     Deletes a callback for the callback list of InitDiscoveryScan.
          Parameters callback (Function) – The callback to delete.
     See also:
     XBeeNetwork.add_init_discovery_scan_callback()
del_network_modified_callback (callback)
```

Deletes a callback for the callback list of NetworkModified.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeNetwork.add network modified callback()
```

# del\_packet\_received\_from\_callback (node, callb=None)

Deletes a received packet callback from the provided node.

#### **Parameters**

- node (RemoteXBeeDevice) The node to listen for frames.
- callb (Function, optional, default=`None`) The callback to delete, None to delete all.

# See also:

```
XBeeNetwork.add_packet_received_from_callback()
```

#### discover device (node id)

Blocking method. Discovers and reports the first remote XBee that matches the supplied identifier.

**Parameters** node\_id (String) – Node identifier of the node to discover.

# Returns

**Discovered remote XBee**, *None* if the timeout expires and the node was not found.

Return type RemoteXBeeDevice

#### See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.set_discovery_timeout()
```

# discover\_devices (device\_id\_list)

Blocking method. Attempts to discover a list of nodes and add them to the current network.

This method does not guarantee that all nodes of *device\_id\_list* will be found, even if they exist physically. This depends on the node discovery operation and timeout.

**Parameters** device\_id\_list (List) – List of device IDs to discover.

## Returns

**List with the discovered nodes. It may not contain all nodes** specified in *device\_id\_list*.

Return type List

# See also:

```
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_options()
XBeeNetwork.get_discovery_timeout()
```

```
XBeeNetwork.set_discovery_timeout()
```

```
export (dir_path=None, name=None, desc=None)
```

Exports this network to the given file path.

If the provided path already exists the file is removed.

#### **Params:**

dir\_path (String, optional, default='None'): Absolute path of the directory to export the network. It should not include the file name. If not defined home directory is used.

name (String, optional, default='None'): Network human readable name. desc (String, optional, default='None'): Network description.

## Returns

**Tuple with result (0: success, 1: failure)** and string (exported file path if success, error string otherwise).

Return type Tuple (Integer, String)

## get\_connections()

Returns a copy of the XBee network connections.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Returns** A copy of the list of *Connection* for the network.

Return type List

See also:

```
XBeeNetwork.get_node_connections()
XBeeNetwork.start_discovery_process()
```

# get\_deep\_discovery\_options()

Returns the deep discovery process options.

## Returns

(NeighborDiscoveryMode, Boolean): Tuple containing:

- mode (NeighborDiscoveryMode): Neighbor discovery mode, the way to perform the network discovery process.
- **remove\_nodes** (**Boolean**): *True* **to remove nodes from the** network if they were not discovered in the last scan, *False* otherwise.

Return type Tuple

See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.set_deep_discovery_timeouts()
```

```
XBeeNetwork.start discovery process()
```

# get\_deep\_discovery\_timeouts()

Gets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see start\_discovery\_process())

#### Returns

# **Tuple containing:**

- node\_timeout (Float): Maximum duration in seconds of the discovery process per node. This is used to find neighbors of a node. This timeout is highly dependent on the nature of the network:
  - It should be greater than the highest 'NT' (Node Discovery Timeout) of your network.
  - And include enough time to let the message propagate depending on the sleep cycle of your network nodes.
- time\_bw\_nodes (Float): Time to wait between node neighbors requests.

  Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float): Time to wait before starting a new network scan.

**Return type** Tuple (Float, Float, Float)

# See also:

```
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

## get\_device\_by\_16(x16bit\_addr)

Returns the XBee in the network whose 16-bit address matches the given one.

Parameters **x16bit\_addr** (XBee16BitAddress) - 16-bit address of the node to retrieve.

**Returns** XBee in the network or *Non* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x16bit\_addr is None or unknown.

# get\_device\_by\_64 (x64bit\_addr)

Returns the XBee in the network whose 64-bit address matches the given one.

**Parameters x64bit\_addr** (XBee64BitAddress) – 64-bit address of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If x64bit\_addr is None or unknown.

#### get\_device\_by\_node\_id (node\_id)

Returns the XBee in the network whose node identifier matches the given one.

Parameters node\_id (String) - Node identifier of the node to retrieve.

**Returns** XBee in the network or *None* if not found.

Return type AbstractXBeeDevice

**Raises** ValueError – If node\_id is None.

# get\_devices()

Returns a copy of the XBee devices list of the network.

If a new XBee node is added to the list after the execution of this method, this new XBee is not added to the list returned by this method.

**Returns** A copy of the XBee devices list of the network.

Return type List

## get\_discovery\_callbacks()

Returns the API callbacks that are used in the device discovery process.

This callbacks notify the user callbacks for each XBee discovered.

#### Returns

**Callback for generic devices discovery** process, callback for discovery specific XBee ops.

**Return type** Tuple (Function, Function)

# get\_discovery\_options()

Returns the network discovery process options.

**Returns** Discovery options value.

Return type Bytearray

#### Raises

- TimeoutException If response is not received before the read timeout expires.
- $\bullet$  XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

# get\_discovery\_timeout()

Returns the network discovery timeout.

Returns Network discovery timeout.

Return type Float

# Raises

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## get\_node\_connections (node)

Returns the network connections with one of their ends *node*.

A deep discover must be performed to get the connections between network nodes.

If a new connection is added to the list after the execution of this method, this new connection is not added to the list returned by this method.

**Parameters node** (AbstractXBeeDevice) – The node to get its connections.

**Returns** List of Connection with node end.

Return type List

See also:

```
XBeeNetwork.get_connections()
XBeeNetwork.start_discovery_process()
```

#### classmethod get\_nt\_limits(protocol)

Returns a tuple with the minimum and maximum values for the 'NT' value depending on the protocol.

#### **Returns**

Minimum value in seconds, maximum value in seconds.

```
Return type Tuple (Float, Float)
```

# get\_number\_devices()

Returns the number of nodes in the network.

**Returns** Number of nodes in the network.

Return type Integer

# has\_devices()

Returns whether there is any device in the network.

# Returns

*True* if there is at least one node in the network, *False* otherwise.

Return type Boolean

# is\_discovery\_running()

Returns whether the discovery process is running.

**Returns** *True* if the discovery process is running, *False* otherwise.

Return type Boolean

# is node in network (node)

Checks if the provided node is in the network or if it is the local XBee.

**Parameters node** (AbstractXBeeDevice) – The node to check.

**Returns** *True* if the node is in the network, *False* otherwise.

## **Return type** Boolean

**Raises** ValueError – If node is None.

# remove\_device (remote\_xbee)

Removes the provided remote XBee from the network.

**Parameters** remote\_xbee (RemoteXBeeDevice) - Remote XBee to remove.

**Raises** ValueError – If the provided *remote xbee* is not in the network.

## scan counter

Returns the scan counter.

Returns The scan counter.

**Return type** Integer

set\_deep\_discovery\_options (deep\_mode=<NeighborDiscoveryMode.CASCADE: (0, 'Cas-cade')>, del\_not\_discovered\_nodes\_in\_last\_scan=False)

Configures the deep discovery options with the given values. These options are only applicable for "de

Configures the deep discovery options with the given values. These options are only applicable for "deep" discovery (see <code>start\_discovery\_process()</code>)

#### **Parameters**

- deep\_mode (NeighborDiscoveryMode, optional, default='NeighborDiscoveryMode.CASCADE') Neighbor discovery mode, the way to perform the network discovery process.
- del\_not\_discovered\_nodes\_in\_last\_scan (Boolean, optional, default=`False`) True to remove nodes from the network if they were not discovered in the last scan.

## See also:

```
digi.xbee.models.mode.NeighborDiscoveryMode
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

#### set\_deep\_discovery\_timeouts (node\_timeout=None,

time\_bw\_requests=None,

time\_bw\_scans=None)
Sets deep discovery network timeouts. These timeouts are only applicable for "deep" discovery (see
start\_discovery\_process())

- **node\_timeout** (**Float, optional, default='None'):** Maximum duration in seconds of the discovery process used to find neighbors of a node. If *None* already configured timeouts are used.
- time\_bw\_requests (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_REQUESTS'): Time to wait between node neighbors requests. It must be between MIN\_TIME\_BETWEEN\_REQUESTS and MAX\_TIME\_BETWEEN\_REQUESTS seconds inclusive. Use this setting not to saturate your network:
  - For 'Cascade', the number of seconds to wait after completion of the neighbor discovery process of the previous node.
  - For 'Flood', the minimum time to wait between each node's neighbor requests.
- time\_bw\_scans (Float, optional, default='DEFAULT\_TIME\_BETWEEN\_SCANS'): Time to wait before starting a new network scan. It must be between MIN\_TIME\_BETWEEN\_SCANS and MAX\_TIME\_BETWEEN\_SCANS seconds inclusive.

**Raises** ValueError – if *node\_timeout*, *time\_bw\_requests* or *time\_bw\_scans* are not between their corresponding limits.

## See also:

```
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.start_discovery_process()
```

## set discovery options(options)

Configures the discovery options (NO parameter) with the given value.

**Parameters options** (Set of *DiscoveryOptions*) – New discovery options, empty set to clear the options.

#### Raises

- ValueError If options is None.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

#### See also:

DiscoveryOptions

# set\_discovery\_timeout (discovery\_timeout)

Sets the discovery network timeout.

**Parameters discovery\_timeout** (Float) - Timeout in seconds.

## Raises

- ValueError If discovery\_timeout is not between the allowed minimum and maximum values.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

## start\_discovery\_process (deep=False, n\_deep\_scans=1)

Starts the discovery process. This method is not blocking.

This process can discover node neighbors and connections, or only nodes:

· Deep discovery: Network nodes and connections between them (including quality) are discovered.

The discovery process will be running the number of scans configured in  $n\_deep\_scans$ . A scan is considered the process of discovering the full network. If there are more than one number of scans configured, after finishing one another is started, until  $n\_deep\_scans$  is satisfied.

See set\_deep\_discovery\_options() to establish the way the network discovery process is performed.

• No deep discovery: Only network nodes are discovered.

The discovery process will be running until the configured timeout expires or, in case of 802.15.4, until the 'end' packet is read.

It may occur that, after timeout expiration, there are nodes that continue sending discovery responses to the local XBee. In this case, these nodes will not be added to the network.

In 802.15.4, both (deep and no deep discovery) are the same and none discover the node connections or their quality. The difference is the possibility of running more than one scan using a deep discovery.

#### **Parameters**

- **deep** (Boolean, optional, default=`False`) True for a deep network scan, looking for neighbors and their connections, False otherwise.
- n\_deep\_scans (Integer, optional, default=1) Number of scans to perform before automatically stopping the discovery process. SCAN\_TIL\_CANCEL means the process will not be automatically stopped. Only applicable if deep=True.

#### See also:

```
XBeeNetwork.add_device_discovered_callback()
XBeeNetwork.add_discovery_process_finished_callback()
XBeeNetwork.del_device_discovered_callback()
XBeeNetwork.del_discovery_process_finished_callback()
XBeeNetwork.get_deep_discovery_options()
XBeeNetwork.set_deep_discovery_options()
XBeeNetwork.get_deep_discovery_timeouts()
XBeeNetwork.set_deep_discovery_timeouts()
XBeeNetwork.get_discovery_options()
XBeeNetwork.set_discovery_options()
XBeeNetwork.get_discovery_timeout()
XBeeNetwork.get_discovery_timeout()
```

# stop\_discovery\_process()

Stops the discovery process if it is running.

Note that some DigiMesh/DigiPoint devices are blocked until the discovery time configured ('NT' parameter) has elapsed, so, when trying to get/set any parameter during the discovery process, a TimeoutException is raised.

```
class digi.xbee.devices.NetworkEventType(code, description)
    Bases: enum.Enum
```

Enumerates the different network event types.

## Values:

**NetworkEventType.ADD** = (0, 'XBee added to the network')

**NetworkEventType.DEL** = (1, 'XBee removed from the network')

**NetworkEventType.UPDATE** = (2, 'XBee in the network updated')

**NetworkEventType.CLEAR** = (3, 'Network cleared')

#### code

Returns the code of the *NetworkEventType* element.

**Returns** Integer: Code of the *NetworkEventType* element.

# description

Returns the description of the *NetworkEventType* element.

**Returns** Description of the *NetworkEventType* element.

Return type String

class digi.xbee.devices.NetworkEventReason(code, description)

Bases: enum. Enum

Enumerates the different network event reasons.

#### Values:

**NetworkEventReason.DISCOVERED** = (0, 'Discovered XBee')

**NetworkEventReason.NEIGHBOR** = (1, 'Discovered as XBee neighbor')

**NetworkEventReason.RECEIVED\_MSG** = (2, 'Received message from XBee')

**NetworkEventReason.MANUAL** = (3, 'Manual modification')

**NetworkEventReason.ROUTE** = (4, 'Hop of a network route')

**NetworkEventReason.READ\_INFO** = (5, 'Read XBee information')

NetworkEventReason.FIRMWARE\_UPDATE = (6, 'The firmware of the device was updated')

#### code

Returns the code of the NetworkEventReason element.

**Returns** Code of the *NetworkEventReason* element.

Return type Integer

## description

Returns the description of the NetworkEventReason element.

**Returns** Description of the *NetworkEventReason* element.

Return type String

**class** digi.xbee.devices.**LinkQuality**(lq=None, is\_rssi=False)

Bases: object

This class represents the link quality of a connection. It can be a LQI (Link Quality Index) for Zigbee devices, or RSSI (Received Signal Strength Indicator) for the rest.

Class constructor. Instantiates a new LinkQuality.

**Parameters** 

- lq(Integer, optional, default=`UNKNOWN`) Link quality.
- is\_rssi (Boolean, optional, default=`False`) True to specify the value is a RSSI, False for LQI.

UNKNOWN = <digi.xbee.devices.LinkQuality object>

Unknown link quality.

#### UNKNOWN VALUE = -9999

Unknown link quality value.

lq

Returns the link quality value.

**Returns** The link quality value.

**Return type** Integer

# is\_rssi

Returns whether this is a RSSI value.

**Returns** *True* if this is an RSSI value, *False* for LQI.

Return type Boolean

class digi.xbee.devices.Connection( $node\_a$ ,  $node\_b$ ,  $lq\_a2b=None$ ,  $lq\_b2a=None$ ,  $status\_a2b=None$ ,  $status\_b2a=None$ )

Bases: object

This class represents a generic connection between two nodes in a XBee network. It contains the source and destination nodes, the link quality of the connection between them and its status.

Class constructor. Instantiates a new Connection.

#### **Parameters**

- node\_a (AbstractXBeeDevice) One of the connection ends.
- node\_b (AbstractXBeeDevice) The other connection end.
- lq\_a2b (LinkQuality or Integer, optional, default='None') Link quality for the connection node\_a -> node\_b. If not specified LinkQuality.UNKNOWN is used.
- lq\_b2a (LinkQuality or Integer, optional, default='None') Link quality for the connection node\_b -> node\_a. If not specified LinkQuality.UNKNOWN is used.
- **status\_a2b** (digi.xbee.models.zdo.RouteStatus, optional, default='None') The status for the connection node\_a -> node\_b. If not specified RouteStatus.UNKNOWN is used.
- status\_b2a (digi.xbee.models.zdo.RouteStatus, optional, default='None') The status for the connection node\_b -> node\_a. If not specified RouteStatus.UNKNOWN is used.

**Raises** ValueError – If *node\_a* or *node\_b* is *None*.

See also:

AbstractXBeeDevice
LinkQuality
digi.xbee.models.zdo.RouteStatus

# node\_a

Returns the node A of this connection.

```
Returns The node A.
           Return type AbstractXBeeDevice
     See also:
     AbstractXBeeDevice
node_b
     Returns the node B of this connection.
           Returns The node B.
           Return type AbstractXBeeDevice
     See also:
     AbstractXBeeDevice
lq_a2b
     Returns the link quality of the connection from node A to node B.
           Returns Link quality for the connection A -> B.
           Return type LinkQuality
     See also:
     LinkQuality
lq_b2a
     Returns the link quality of the connection from node B to node A.
           Returns Link quality for the connection B \rightarrow A.
           Return type LinkQuality
     See also:
     LinkQuality
status_a2b
     Returns the status of this connection from node A to node B.
           Returns The status for A -> B connection.
           Return type RouteStatus
     See also:
     digi.xbee.models.zdo.RouteStatus
```

#### status b2a

Returns the status of this connection from node B to node A.

**Returns** The status for B -> A connection.

Return type RouteStatus

See also:

```
digi.xbee.models.zdo.RouteStatus
```

#### scan counter a2b

Returns the scan counter for this connection, discovered by its A node.

**Returns** The scan counter for this connection, discovered by its A node.

Return type Integer

## scan\_counter\_b2a

Returns the scan counter for this connection, discovered by its B node.

Returns The scan counter for this connection, discovered by its B node.

Return type Integer

# digi.xbee.exception module

```
exception digi.xbee.exception.XBeeException
```

Bases: Exception

Generic XBee API exception. This class and its subclasses indicate conditions that an application might want to catch.

All functionality of this class is the inherited of Exception.

#### with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

# exception digi.xbee.exception.CommunicationException

```
Bases: digi.xbee.exception.XBeeException
```

This exception will be thrown when any problem related to the communication with the XBee device occurs.

All functionality of this class is the inherited of Exception.

## with traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

```
exception digi.xbee.exception.ATCommandException (message='There was a problem sending the AT command packet.', cmd_status=None)
```

 $Bases: \ \textit{digi.xbee.exception.CommunicationException}$ 

This exception will be thrown when a response of a packet is not success or OK.

All functionality of this class is the inherited of Exception.

# with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

```
Bases: digi.xbee.exception.XBeeException
     This exception will be thrown when any problem related to the connection with the XBee device occurs.
     All functionality of this class is the inherited of Exception.
     with traceback()
          Exception.with traceback(tb) – set self. traceback to tb and return self.
exception digi.xbee.exception.XBeeDeviceException
     Bases: digi.xbee.exception.XBeeException
     This exception will be thrown when any problem related to the XBee device occurs.
     All functionality of this class is the inherited of Exception.
     with traceback()
          Exception.with_traceback(tb) - set self.__traceback__ to tb and return self.
exception digi.xbee.exception.InvalidConfigurationException (message='The con-
                                                                              figuration used to
                                                                              open the interface is
                                                                              invalid.')
     Bases: digi.xbee.exception.ConnectionException
     This exception will be thrown when trying to open an interface with an invalid configuration.
     All functionality of this class is the inherited of Exception.
     with_traceback()
          Exception.with_traceback(tb) - set self.__traceback__ to tb and return self.
exception digi.xbee.exception.InvalidOperatingModeException (message=None,
                                                                              op_mode=None)
     Bases: digi.xbee.exception.ConnectionException
     This exception will be thrown if the operating mode is different than OperatingMode.API_MODE and Operat-
     ingMode.API_MODE
     All functionality of this class is the inherited of Exception.
     with traceback()
          Exception.with_traceback(tb) - set self.__traceback__ to tb and return self.
exception digi.xbee.exception.InvalidPacketException (message='The
                                                                                     XBee
                                                                                             API
                                                                     packet is not properly formed.')
     Bases: digi.xbee.exception.CommunicationException
     This exception will be thrown when there is an error parsing an API packet from the input stream.
     All functionality of this class is the inherited of Exception.
     with traceback()
          Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
exception digi.xbee.exception.OperationNotSupportedException (message='The re-
                                                                                quested operation
                                                                                is not supported by
                                                                                either the connec-
                                                                                tion interface or
                                                                               the XBee device.')
     Bases: digi.xbee.exception.XBeeDeviceException
```

exception digi.xbee.exception.ConnectionException

2.6. API reference 935

This exception will be thrown when the operation performed is not supported by the XBee device.

All functionality of this class is the inherited of Exception.

#### with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

**exception** digi.xbee.exception.**TimeoutException** (message='There was a timeout while executing the requested operation.')

Bases: digi.xbee.exception.CommunicationException

This exception will be thrown when performing synchronous operations and the configured time expires.

All functionality of this class is the inherited of Exception.

# with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

**exception** digi.xbee.exception.**TransmitException** (message='There was a problem with a transmitted packet response (status not ok)', transmit status=None)

Bases: digi.xbee.exception.CommunicationException

This exception will be thrown when receiving a transmit status different than *TransmitStatus.SUCCESS* after sending an XBee API packet.

All functionality of this class is the inherited of Exception.

# with\_traceback()

Exception.with traceback(tb) – set self. traceback to tb and return self.

**exception** digi.xbee.exception.**XBeeSocketException**(*message='There was a socket er-ror'*, *status=None*)

Bases: digi.xbee.exception.XBeeException

This exception will be thrown when there is an error performing any socket operation.

All functionality of this class is the inherited of Exception.

# with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

# $\textbf{exception} \ \, \texttt{digi.xbee.exception.} \textbf{FirmwareUpdateException}$

Bases: digi.xbee.exception.XBeeException

This exception will be thrown when any problem related to the firmware update process of the XBee device occurs.

All functionality of this class is the inherited of Exception.

## with traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

# exception digi.xbee.exception.RecoveryException

Bases: digi.xbee.exception.XBeeException

This exception will be thrown when any problem related to the auto-recovery process of the XBee device occurs.

All functionality of this class is the inherited of Exception.

## with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

# digi.xbee.filesystem module

Bases: object

Class used to represent XBee file system elements (files and directories).

Class constructor. Instantiates a new FileSystemElement object with the given parameters.

#### **Parameters**

- name (String or bytearray) Name of the file system element.
- path (String or bytearray, optional, default=`None`) Absolute path of the element.
- is\_dir (Boolean, optional, default=`True`) True if the element is a directory, False for a file.
- **size** (Integer, optional, default=0) Element size in bytes. Only for files.
- is\_secure (Boolean, optional, default=`False`) True for a secure element, False otherwise.

Raises ValueError – If any of the parameters are invalid.

#### name

Returns the file system element name.

**Returns** File system element name.

Return type String

#### path

Returns the file system element absolute path.

**Returns** File system element absolute path.

Return type String

# is\_dir

Returns whether the file system element is a directory.

**Returns** *True* for a directory, *False* otherwise.

Return type Boolean

# size

Returns the size in bytes of the element.

**Returns** The size in bytes of the file, 0 for a directory.

**Return type** Integer

# size\_pretty

Returns a human readable size (e.g., 1K 234M 2G).

Returns Human readable size.

Return type String

## is secure

Returns whether the element is secure.

**Returns** *True* for a secure element, *False* otherwise.

## Return type Boolean

#### static from\_data (name, size, flags, path=None)

Creates a file element from its name and the bytearray with info and size.

#### **Parameters**

- name (String or bytearray) The name of the element to create.
- **size** (*Bytearray*) Byte array containing file size.
- **flags** (Integer) Integer with file system element information.
- path(String or bytearray, optional, default=`None`)—The absolute path of the element (without its name).

**Returns** The new file system element.

Return type FileSystemElement

```
exception digi.xbee.filesystem.FileSystemException(message, fs_status=None)
```

Bases: digi.xbee.exception.XBeeException

This exception will be thrown when any problem related with the XBee file system occurs.

All functionality of this class is the inherited from Exception.

# with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

```
\textbf{exception} \ \texttt{digi.xbee.filesystem.FileSystemNotSupportedException} \ (\textit{message},
```

 $fs\_status=None$ )

Bases: digi.xbee.filesystem.FileSystemException

This exception will be thrown when the file system feature is not supported in the device.

All functionality of this class is the inherited from Exception.

# with\_traceback()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

```
{\tt class} \ {\tt digi.xbee.filesystem.FileProcess} \ (\textit{f\_mng}, \textit{file}, \textit{timeout})
```

Bases: object

This class represents a file process.

Class constructor. Instantiates a new \_FileProcess object with the provided parameters.

# **Parameters**

- (class  $(f_mng)$  .File System Manager): The file system manager.
- **file** (FileSystemElement or String) File or its absolute path.
- timeout (Float) Timeout in seconds.

#### running

Returns if this file command is running.

**Returns** *True* if it is running, *False* otherwise.

Return type Boolean

#### status

Returns the status code.

**Returns** The status.

Return type Integer

#### block size

Returns the size of the block for this file operation.

**Returns** Size of the block for this file operation.

Return type Integer

# class digi.xbee.filesystem.FileSystemManager(xbee)

Bases: object

Helper class used to manage local or remote XBee file system.

Class constructor. Instantiates a new FileSystemManager with the given parameters.

**Parameters** xbee (AbstractXBeeDevice) – XBee to manage its file system.

Raises FileSystemNotSupportedException - If the XBee does not support filesystem.

#### xbee

Returns the XBee of this file system manager.

Returns XBee to manage its file system.

Return type AbstractXBeeDevice

#### np\_value

The 'NP' parameter value of the local XBee.

**Returns** The 'NP' value.

Return type Integer

## get\_root()

Returns the root directory.

**Returns** The root directory.

Return type FileSystemElement

**Raises** FileSystemException – If there is any error performing the operation or the function is not supported.

make\_directory (dir\_path, base=None, mk\_parents=True, timeout=20)

Creates the provided directory.

## **Parameters**

- **dir\_path** (*String*) Path of the new directory to create. It is relative to the directory specify in base.
- base (FileSystemElement, optional, default='None) Base directory. If not specify it refers to '/flash'.
- mk\_parents (Boolean, optional, default=`True`) True to make parent directories as needed, False otherwise.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion. If mk\_parents this is the timeout per directory creation.

**Returns** List of FileSystemElement created directories.

Return type List

## Raises

• FileSystemException – If there is any error performing the operation or the function is not supported.

• ValueError – If any of the parameters is invalid.

# list\_directory (directory=None, timeout=20)

Lists the contents of the given directory.

#### **Parameters**

- **directory** (FileSystemElement or String) Directory to list or its absolute path.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

#### Returns

**List of** *:class:***.FilesystemElement' objects contained in** the given directory, empty list if status is not 0.

# Return type List

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

**remove** (*entry*, *rm\_children=True*, *timeout=20*)

Removes the given file system entry.

All files in a directory must be deleted before removing the directory. On XBee 3 802.15.4, DigiMesh, and Zigbee, deleted files are marked as unusable space unless they are at the "end" of the file system (most-recently created). On these products, deleting a file triggers recovery of any deleted file space at the end of the file system, and can lead to a delayed response.

## **Parameters**

- **entry** (FileSystemElement or String) File system entry to remove or its absolute path.
- rm\_children (Boolean, optional, default=`True`) True to remove directory children if they exist, False otherwise.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

# read\_file (file, offset=0, progress\_cb=None)

Reads from the provided file starting at the given offset. If there is no progress callback the function blocks until the required amount of bytes is read.

## **Parameters**

- **file** (FileSystemElement or String) File to read or its absolute path.
- offset (Integer, optional, default=0) File offset to start reading.
- progress\_cb (Function, optional, default=`None`) Function called when new data is read. Receives four arguments:

- The chunk of data read as byte array.
- The progress percentage as float.
- The total size of the file.
- The status when process finishes.

**Returns** The process to read data from the file.

Return type FileProcess

#### Raises

- FileSystemException If there is any error performing the operation and progress\_cb is None.
- ValueError If any of the parameters is invalid.

#### See also:

```
get_file()
```

# write\_file (file, offset=0, secure=False, options=None, progress\_cb=None)

Writes to the provided file the data starting at the given offset. The function blocks until the all data is written.

#### **Parameters**

- **file** (FileSystemElement or String) File to write or its absolute path.
- offset (Integer, optional, default=0) File offset to start writing.
- **secure** (Boolean, optional, default=`False`) *True* to store the file securely (no read access), *False* otherwise.
- **options** (*Dictionary*, *optional*) Other write options as list: *exclusive*, *truncate*, *append*.
- progress\_cb (Function, optional, default=`None`) Function call when data is written. Receives three arguments:
  - The amount of bytes written (for each chunk).
  - The progress percentage as float.
  - The status when process finishes.

# Raises

- FileSystemException If there is any error performing the operation and progress\_cb is None.
- ValueError If any of the parameters is invalid.

#### See also:

```
put_file()
```

# get\_file (src, dest, progress\_cb=None)

Downloads the given XBee file in the specified destination path.

#### **Parameters**

- **src** (FileSystemElement or String) File to download or its absolute path.
- **dest** (*String*) The absolute path of the destination file.
- progress\_cb (Function, optional) Function call when data is being downloaded. Receives three arguments:
  - The progress percentage as float.
  - Destination file path.
  - Source file path.

#### Raises

- FileSystemException If there is any error performing the operation and progress\_cb is None.
- ValueError If any of the parameters is invalid.

**put\_file** (*src*, *dest*, *secure=False*, *overwrite=False*, *mk\_parents=True*, *progress\_cb=None*) Uploads the given file to the specified destination path of the XBee.

#### **Parameters**

- **src** (*String*) Absolute path of the file to upload.
- **dest** (FileSystemElement or String) The file in the XBee or its absolute path.
- **secure** (Boolean, optional, default=`False`) *True* if the file should be stored securely, *False* otherwise.
- **overwrite** (Boolean, optional, default=`False`) *True* to overwrite the file if it exists, *False* otherwise.
- mk\_parents (Boolean, optional, default=`True`) True to make parent directories as needed, False otherwise.
- progress\_cb (Function, optional) Function call when data is being uploaded. Receives two arguments:
  - The progress percentage as float.
  - Destination file path.
  - Source file path.

Returns The new created file.

Return type FileSystemElement

#### Raises

- FileSystemException If there is any error performing the operation and progress\_cb is None.
- ValueError If any of the parameters is invalid.

put\_dir (src, dest='/flash', verify=True, progress\_cb=None)

Uploads the given source directory contents into the given destination directory in the XBee.

# **Parameters**

• **src** (*String*) – Local directory to upload its contents.

- **dest** (FileSystemElement or String) The destination dir in the XBee or its absolute path. Defaults to '/flash'.
- **verify** (Boolean, optional, default=`True`) True to check the hash of the uploaded content.
- progress\_cb (Function, optional) Function call when data is being uploaded. Receives three argument:
  - The progress percentage as float.
  - Destination file path.
  - The absolute path of the local being uploaded as string.

#### Raises

- FileSystemException If there is any error performing the operation and progress\_cb is None.
- ValueError If any of the parameters is invalid.

### get file hash(file, timeout=20)

Returns the SHA256 hash of the given file.

#### **Parameters**

- **file** (FileSystemElement or String) File to get its hash or its absolute path.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

**Returns** SHA256 hash of the given file.

**Return type** Bytearray

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

# move (source, dest, timeout=20)

Moves the given source element to the given destination path.

#### **Parameters**

- **source** (FileSystemElement or String) Source entry to move.
- **dest** (*String*) Destination path of the element to move.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

# get\_volume\_info(vol='/flash', timeout=20)

Returns the file system volume information. Currently '/flash' is the only supported value.

#### **Parameters**

- vol (FileSystemElement`or String, optional, default=/flash') - Volume name.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

**Returns** Collection of pair values describing volume information.

Return type Dictionary

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

#### See also:

*FSCommandStatus* 

#### format (vol='/flash', timeout=30)

Formats provided volume. Currently '/flash' is the only supported value. Formatting the file system takes time, and any other requests will fail until it completes and sends a response.

#### **Parameters**

- vol (FileSystemElement`or String, optional, default=/flash') - Volume name.
- timeout(Float, optional, default=`DEFAULT\_FORMAT\_TIMEOUT`)

   Maximum number Of seconds to wait for the operation completion.

**Returns** Collection of pair values describing volume information.

Return type Dictionary

## Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

#### See also:

*FSCommandStatus* 

# pget\_path\_id (dir\_path, path\_id=0, timeout=20)

Returns the directory path id of the given path. Returned directory path id expires if not referenced in 2 minutes.

#### **Parameters**

- dir\_path (String) Path of the directory to get its id. It is relative to the directory path id.
- path\_id (Integer, optional, default=0) Directory path id. 0 for the root directory.

• **timeout** (*Float*, *optional*, *default=`DEFAULT\_TIMEOUT`*) – Maximum number of seconds to wait for the operation completion.

#### Returns

**Status of the file system command** execution, new directory path id (-1 if status is not 0) and its absolute path (empty if status is not 0). The full path may be *None* or empty if it is too long and exceeds the communication frames length.

**Return type** Tuple (Integer, Integer, String)

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

#### See also:

FSCommandStatus 5 4 1

# pmake\_directory (dir\_path, path\_id=0, timeout=20)

Creates the provided directory. Parent directories of the one to be created must exist. Separate requests must be dane to make intermediate directories.

#### **Parameters**

- **dir\_path** (*String*) Path of the new directory to create. It is relative to the directory path id.
- path\_id (Integer, optional, default=0) Directory path id. 0 for the root directory.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion. If mk\_parents this is the timeout per directory creation.

# Returns

Status of the file system command execution (see FSCommandStatus).

# Return type Integer

## Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

### See also:

**FSCommandStatus** 

# plist\_directory (dir\_path, path\_id=0, timeout=20)

Lists the contents of the given directory.

#### **Parameters**

- dir\_path (String) Path of the directory to list. It is relative to the directory path id.
- path\_id (Integer, optional, default=0) Directory path id. 0 for the root directory.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

#### Returns

**Status of the file system command execution** and a list of *:class:*.FilesystemElement' objects contained in the given directory, empty list if status is not 0.

Return type Tuple (Integer, List)

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

#### See also:

FSCommandStatus

premove (entry\_path, path\_id=0, timeout=20)

Removes the given file system entry.

All files in a directory must be deleted before removing the directory. On XBee 3 802.15.4, DigiMesh, and Zigbee, deleted files are marked as as unusable space unless they are at the "end" of the file system (most-recently created). On these products, deleting a file triggers recovery of any deleted file space at the end of the file system, and can lead to a delayed response.

## **Parameters**

- **entry\_path** (*String*) Path of the entry to remove. It is relative to the directory path id.
- path\_id (Integer, optional, default=0) Directory path id. 0 for the root directory.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

#### Returns

Status of the file system command execution (see FSCommandStatus).

Return type Integer

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

## See also:

**FSCommandStatus** 

# popen\_file (file\_path, path\_id=0, options=<FileOpenRequestOption.READ: 4>, timeout=20)

Open a file for reading and/or writing. Use the *FileOpenRequestOption.SECURE* (0x80) bitmask for options to upload a write-only file (one that cannot be downloaded or viewed), useful for protecting files on the device. Returned file id expires if not referenced in 2 minutes.

#### **Parameters**

- **file\_path** (*String*) Path of the file to open. It is relative to the directory path id.
- path\_id (Integer, optional, default=0) Directory path id. 0 for the root directory.
- options(Integer, optional, default=`FileOpenRequestOption. READ`) — Bitmask that specifies the options to open the file. It defaults to FileOpenRequestOption.READ which means open for reading. See FileOpenRequestOption for more options.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

#### **Returns**

**Status of the file system** command execution (see *FSCommandStatus*), the file id to use in later requests, and the size of the file (in bytes), 0xFFFFFFFF if unknown.

Return type Tuple (Integer, Integer, Integer)

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

### See also:

```
FileOpenRequestOption
FSCommandStatus
pclose_file()
```

# ${\tt pclose\_file}~(\mathit{file\_id}, \mathit{timeout}{=}20)$

Closes an open file and releases its file handle.

## **Parameters**

- **file\_id** (*Integer*) File id returned when opening.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

#### **Returns**

Status of the file system command execution (see FSCommandStatus).

Return type Integer

Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

#### See also:

```
FSCommandStatus
popen_file()
```

# pread\_file (file\_id, offset=-1, size=-1, timeout=20)

Reads from the provided file the given amount of bytes starting at the given offset. The file must be opened for reading first.

## **Parameters**

- **file\_id** (*Integer*) File id returned when opening.
- **offset** (Integer, optional, default=-1) File offset to start reading. -1 to use current position.
- **size**(Integer, optional, default=-1) Number of bytes to read. -1 to read as many as possible.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

## Returns

Status of the file system command execution (see FSCommandStatus), the file id, the offset of the read data, and the read data.

**Return type** Tuple (Integer, Integer, Integer, Bytearray)

## **Raises**

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

# See also:

```
FSCommandStatus
popen_file()
```

## pwrite\_file (file\_id, data, offset=-1, timeout=20)

Writes to the provided file the given data bytes starting at the given offset. The file must be opened for writing first.

#### **Parameters**

- **file id** (*Integer*) File id returned when opening.
- data (Bytearray, bytes or String) Data to write.
- **offset** (Integer, optional, default=-1) File offset to start writing. -1 to use current position.

• **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) — Maximum number of seconds to wait for the operation completion.

#### Returns

**Status of the file system** command execution (see *FSCommandStatus*), the file id, and the current offset after writing.

Return type Tuple (Integer, Integer, Integer)

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

#### See also:

```
FSCommandStatus
popen_file()
```

```
pget_file_hash (file_path, path_id=0, timeout=20)
```

Returns the SHA256 hash of the given file.

#### **Parameters**

- **file\_path** (String) Path of the file to get its hash. It is relative to the directory path id.
- path\_id (Integer, optional, default=0) Directory path id. 0 for the root directory.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

# Returns

**Status of the file system command** execution and SHA256 hash of the given file (empty bytearray if status is not 0).

Return type Tuple (Integer, Bytearray)

# Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

## See also:

FSCommandStatus

**prename** (*current\_path*, *new\_path*, *path\_id=0*, *timeout=20*)
Rename provided file.

#### **Parameters**

- **current\_path** (*String*) Current path name. It is relative to the directory path id.
- **new\_path** (*String*) New name. It is relative to the directory path id.
- path\_id (Integer, optional, default=0) Directory path id. 0 for the root directory.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

## Returns

Status of the file system command execution (see FSCommandStatus).

Return type Integer

## Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

#### See also:

FSCommandStatus

# prelease\_path\_id (path\_id, timeout=20)

Releases the provided directory path id.

# **Parameters**

- path\_id (Integer) Directory path id to release.
- **timeout** (Float, optional, default=`DEFAULT\_TIMEOUT`) Maximum number of seconds to wait for the operation completion.

**Returns** Status of the file system command execution.

Return type Integer

#### Raises

- FileSystemException If there is any error performing the operation or the function is not supported.
- ValueError If any of the parameters is invalid.

## See also:

**FSCommandStatus** 

# class digi.xbee.filesystem.LocalXBeeFileSystemManager(xbee\_device) Bases: object

Helper class used to manage the local XBee file system.

Class constructor. Instantiates a new LocalXBeeFileSystemManager with the given parameters.

**Parameters** xbee\_device (XBeeDevice) - The local XBee to manage its file system.

#### is connected

Returns whether the file system manager is connected or not.

#### Returns

True if the file system manager is connected, False otherwise.

## **Return type** Boolean

#### connect()

Connects the file system manager.

#### Raises

- FileSystemException If there is any error connecting the file system manager.
- FileSystemNotSupportedException If the device does not support filesystem feature.

# disconnect()

Disconnects the file system manager and restores the device connection.

**Raises** XBeeException – If there is any error restoring the XBee connection.

# get\_current\_directory()

Returns the current device directory.

**Returns** Current device directory.

**Return type** String

**Raises** FileSystemException – If there is any error getting the current directory or the function is not supported.

# change\_directory (directory)

Changes the current device working directory to the given one.

**Parameters directory** (String) – New directory to change to.

**Returns** Current device working directory after the directory change.

**Return type** String

**Raises** FileSystemException – If there is any error changing the current directory or the function is not supported.

# make\_directory (directory)

Creates the provided directory.

**Parameters directory** (*String*) – New directory to create.

**Raises** FileSystemException – If there is any error creating the directory or the function is not supported.

# list\_directory (directory=None)

Lists the contents of the given directory.

**Parameters directory** (*String*, *optional*) – the directory to list its contents. If not provided, the current directory contents are listed.

# Returns

**list of** :class:.FilesystemElement' objects contained in the given (or current) directory.

**Return type** List

**Raises** FileSystemException – if there is any error listing the directory contents or the function is not supported.

# remove\_element (element\_path)

Removes the given file system element path.

**Parameters element\_path** (String) – Path of the file system element to remove.

**Raises** FileSystemException – If there is any error removing the element or the function is not supported.

# move\_element (source\_path, dest\_path)

Moves the given source element to the given destination path.

#### **Parameters**

- **source\_path** (*String*) Source path of the element to move.
- **dest\_path** (*String*) Destination path of the element to move.

**Raises** FileSystemException – If there is any error moving the element or the function is not supported.

put\_file (source\_path, dest\_path, secure=False, progress\_callback=None)

Transfers the given file in the specified destination path of the XBee.

#### **Parameters**

- **source\_path** (*String*) the path of the file to transfer.
- **dest\_path** (*String*) the destination path to put the file in.
- **secure** (Boolean, optional, default=`False`) *True* if the file should be stored securely, *False* otherwise.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Takes the following arguments:
  - The progress percentage as integer.

**Raises** FileSystemException – If there is any error transferring the file or the function is not supported.

put\_dir (source\_dir, dest\_dir=None, progress\_callback=None)

Uploads the given source directory contents into the given destination directory in the device.

#### **Parameters**

- **source\_dir** (*String*) Local directory to upload its contents.
- **dest\_dir** (*String*, *optional*) Remote directory to upload the contents to. Defaults to current directory.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Takes the following arguments:
  - The file being uploaded as string.
  - The progress percentage as integer.

**Raises** FileSystemException – If there is any error uploading the directory or the function is not supported.

get\_file (source\_path, dest\_path, progress\_callback=None)

Downloads the given XBee device file in the specified destination path.

#### **Parameters**

952

- **source\_path** (*String*) Path of the XBee device file to download.
- **dest\_path** (String) Destination path to store the file in.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Takes the following arguments:
  - The progress percentage as integer.

**Raises** FileSystemException – If there is any error downloading the file or the function is not supported.

## format\_filesystem()

Formats the device file system.

**Raises** FileSystemException – If there is any error formatting the file system.

#### get\_usage\_information()

Returns the file system usage information.

**Returns** Collection of pair values describing the usage information.

Return type Dictionary

Raises FileSystemException - If there is any error retrieving the file system usage information.

# get\_file\_hash(file\_path)

Returns the SHA256 hash of the given file path.

**Parameters** file\_path (String) - Path of the file to get its hash.

Returns SHA256 hash of the given file path.

**Return type** String

**Raises** FileSystemException – If there is any error retrieving the file hash.

Performs a remote filesystem update operation in the given target.

## **Parameters**

- remote\_device (RemoteXBeeDevice) Remote XBee to update its filesystem image.
- ota\_filesystem\_file (String) Path of the OTA filesystem file to upload.
- max\_block\_size (Integer, optional) Maximum size of the ota block to send.
- timeout (Integer, optional) Timeout to wait for remote frame requests.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

# Raises

- FileSystemNotSupportedException If the target does not support filesystem update.
- FileSystemException If there is any error updating the remote filesystem image.

digi.xbee.filesystem.check\_fs\_support (xbee, min\_fw\_vers=None, max\_fw\_vers=None) Checks if filesystem API feature is supported.

#### **Parameters**

- **xbee** (:AbstractXBeeDevice) The XBee to check.
- min\_fw\_vers (Dictionary, optional, default=`None`) A dictionary with protocol as key, and minimum firmware version with filesystem support as value.
- max\_fw\_vers (Dictionary, optional, default=`None`) A dictionary with protocol as key, and maximum firmware version with filesystem support as value.

**Returns** *True* if filesystem is supported, *False* otherwise.

Return type Boolean

digi.xbee.filesystem.get\_local\_file\_hash(local\_path)

Returns the SHA256 hash of the given local file.

**Parameters** local\_path (*String*) – Absolute path of the file to get its hash.

**Returns** SHA256 hash of the given file.

Return type Bytearray

# digi.xbee.firmware module

class digi.xbee.firmware.UpdateConfigurer(node, timeout=None, callback=None)
 Bases: object

For internal use only. Helper class used to prepare nodes and/or network for an update.

Class constructor. Instantiates a new *UpdateConfigurer* with the given parameters.

## **Parameters**

- node (AbstractXBeeDevice) Target being updated.
- timeout (Float, optional, default=`None`) Operations timeout.
- callback (Function) Function to notify about the progress.

#### sync\_sleep

Returns whether node is part of a DigiMesh synchronous sleeping network.

**Returns** *True* if it synchronous sleeps, *False* otherwise.

Return type Boolean

### prepare\_total

Returns the total work for update preparation step.

Returns Total prepare work.

Return type Integer

## restore total

Returns the total work for update restoration step.

**Returns** Total restore work.

**Return type** Integer

prepare\_for\_update (prepare\_node=True, prepare\_net=True, restore\_later=True)

Prepares the node for an update process.

#### **Parameters**

- prepare\_node (Boolean, optional, default=`True`) True to prepare the node.
- prepare\_net(Boolean, optional, default=`True`) True to prepare the network.
- restore\_later (Boolean, optional, default=`True`) True to restore node original values when finish the update process.

restore\_after\_update (restore\_settings=True, port\_settings=None)

Restores the node after an update process.

#### **Parameters**

- restore\_settings (Boolean, optional, default=`True`) *True* to restore stored settings, *False* otherwise.
- port\_settings (Dictionary, optional, default=`None`) Dictionary with the new serial port configuration, None for remote node or if the serial config has not changed.

**static exec\_at\_cmd** (func, node, cmd, value=None, retries=5, apply=False)
Reads the given parameter from the XBee with the given number of retries.

# **Parameters**

- **func** (Function) Function to execute.
- node (AbstractXBeeDevice) XBee to get/set parameter.
- (String or (cmd) class: ATStringCommand): Parameter to get/set.
- value (Bytearray, optional, default=`None`) Value to set.
- retries (Integer, optional, default=5) Number of retries to perform.
- apply (Boolean, optional, default=`False`) True to apply.

Returns Read parameter value.

**Return type** Bytearray

**Raises** XBeeException – If the value could be get/set after the retries.

# progress\_cb (task, done=0)

If a callback was provided in the constructor, notifies it with the provided task and the corresponding percentage.

### **Parameters**

- task (String) The task to inform about, it must be TASK\_PREPARE or TASK\_RESTORE.
- **done** (Integer, optional, default=0) Total amount of done job. If 0, it is increased by one.

**Returns** Total work done for the task.

## Return type Integer

Performs a local firmware update operation in the given target.

#### **Parameters**

- **target** (String or *XBeeDevice*) Target of the firmware upload operation. String: serial port identifier. *XBeeDevice*: XBee to upload its firmware.
- xml fw file (String) Path of the XML file that describes the firmware.
- **xbee\_firmware\_file** (String, optional) Location of the XBee binary firmware file.
- bootloader\_firmware\_file (String, optional) Location of the bootloader binary firmware file.
- timeout (Integer, optional) Serial port read data timeout.
- progress\_callback (Function, optional) Function to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

Raises FirmwareUpdateException – If there is any error performing the firmware update.

Performs a remote firmware update operation in the given target.

#### **Parameters**

- remote (RemoteXBeeDevice) Remote XBee to upload.
- **xml\_fw\_file** (*String*) Path of the XML file that describes the firmware.
- firmware\_file (String, optional) Path of the binary firmware file.
- bootloader\_file (String, optional) Path of the bootloader firmware file.
- max\_block\_size (Integer, optional) Maximum size of the ota block to send.
- timeout (Integer, optional) Timeout to wait for remote frame requests.
- progress\_callback (Function, optional) Function to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

Raises FirmwareUpdateException — if there is any error performing the remote firmware update.

```
digi.xbee.firmware.update_remote_filesystem(remote, ota_fs_file, max_block_size=0, time-
out=None, progress_callback=None, _pre-
pare=True)
```

Performs a remote filesystem update operation in the given target.

#### **Parameters**

- remote (RemoteXBeeDevice) Remote XBee to update its filesystem.
- ota\_fs\_file (String) Path of the OTA filesystem image file.
- max\_block\_size (Integer, optional) Maximum size of the ota block to send.
- timeout (Integer, optional) Timeout to wait for remote frame requests.
- progress\_callback (Function, optional) Function to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

Raises FirmwareUpdateException - If there is any error updating the remote filesystem image.

# digi.xbee.io module

```
class digi.xbee.io.IOLine (description, index, at_command, pwm_command=None)
    Bases: enum.Enum
```

Enumerates the different IO lines that can be found in the XBee devices.

Depending on the hardware and firmware of the device, the number of lines that can be used as well as their functionality may vary. Refer to the product manual to learn more about the IO lines of your XBee device.

Values:

```
IOLine.DIO0_AD0 = ('DIO0/AD0', 0, 'D0')
IOLine.DIO1_AD1 = ('DIO1/AD1', 1, 'D1')
IOLine.DIO2_AD2 = ('DIO2/AD2', 2, 'D2')
IOLine.DIO3_AD3 = ('DIO3/AD3', 3, 'D3')
IOLine.DIO4_AD4 = ('DIO4/AD4', 4, 'D4')
IOLine.DIO5_AD5 = ('DIO5/AD5', 5, 'D5')
IOLine.DIO6 = ('DIO6', 6, 'D6')
IOLine.DIO7 = ('DIO7', 7, 'D7')
IOLine.DIO8 = ('DIO8', 8, 'D8')
IOLine.DIO9 = ('DIO9', 9, 'D9')
IOLine.DIO10_PWM0 = ('DIO10/PWM0', 10, 'P0', 'M0')
IOLine.DIO11_PWM1 = ('DIO11/PWM1', 11, 'P1', 'M1')
IOLine.DIO12 = ('DIO12', 12, 'P2')
IOLine.DIO13 = ('DIO13', 13, 'P3')
IOLine.DIO14 = ('DIO14', 14, 'P4')
IOLine.DIO15 = ('DIO15', 15, 'P5')
IOLine.DIO16 = ('DIO16', 16, 'P6')
IOLine.DIO17 = ('DIO17', 17, 'P7')
IOLine.DIO18 = ('DIO18', 18, 'P8')
IOLine.DIO19 = ('DIO19', 19, 'P9')
```

#### description

Returns the description of the IOLine element.

**Returns** The description of the IOLine element.

Return type String

#### index

Returns the index of the IOLine element.

**Returns** The index of the IOLine element.

**Return type** Integer

#### at\_command

Returns the AT command of the IOLine element.

**Returns** The AT command of the IOLine element.

Return type String

## pwm\_at\_command

Returns the PWM AT command associated to the IOLine element.

#### Returns

The PWM AT command associated to the IO line, *None* if the IO line does not have a PWM AT command associated.

Return type String

# has\_pwm\_capability()

Returns whether the IO line has PWM capability or not.

**Returns** *True* if the IO line has PWM capability, *False* otherwise.

Return type Boolean

```
class digi.xbee.io.IOValue(code)
```

Bases: enum. Enum

Enumerates the possible values of a *IOLine* configured as digital I/O.

Values:

IOValue.LOW = 4 IOValue.HIGH = 5

code

Returns the code of the IOValue element.

**Returns** The code of the IOValue element.

Return type String

```
class digi.xbee.io.IOSample(io_sample_payload)
```

Bases: object

This class represents an IO Data Sample. The sample is built using the the constructor. The sample contains an analog and digital mask indicating which IO lines are configured with that functionality.

Depending on the protocol the XBee device is executing, the digital and analog masks are retrieved in separated bytes (2 bytes for the digital mask and 1 for the analog mask) or merged contained (digital and analog masks are contained in 2 bytes).

Digital and analog channels masks Indicates which digital and ADC IO lines are configured in the module. Each bit corresponds to one digital or ADC IO line on the module:

```
bit 0 = DIO01
bit 1 =
       DIO10
bit 2 = DIO20
bit 3 = DIO31
bit 4 = DIO40
bit 5 = DIO51
bit 6 = DIO60
bit 7 = DIO70
bit 8 = DIO80
bit 9 = AD00
bit 10 = AD11
bit 11 = AD21
bit 12 = AD30
bit 13 = AD40
bit 14 = AD50
bit 15 = NA0
Example: mask of 0x0C29 means DIO0, DIO3, DIO5, AD1 and AD2 enabled.
```

Digital Channel Mask Indicates which digital IO lines are configured in the module. Each bit corresponds to one digital IO line on the module:

```
bit 0 = DIOOADO
bit 1 =
       DIO1AD1
bit 2 = DIO2AD2
bit 3 = DIO3AD3
bit 4 = DIO4AD4
bit 5 = DIO5AD5ASSOC
bit 6 = DIO6RTS
bit 7 = DIO7CTS
bit 8 = DIO8DTRSLEEP_RQ
bit 9 = DIO9ON_SLEEP
bit 10 = DIO10PWM0RSSI
bit 11 = DIO11PWM1
bit 12 = DIO12CD
bit 13 = DIO13
bit 14 = DIO14
bit 15 = NA
Example: mask of 0x040B means DIO0, DIO1, DIO2, DIO3 and DIO10 enabled.
```

Analog Channel Mask Indicates which lines are configured as ADC. Each bit in the analog channel mask corresponds to one ADC line on the module.

```
bit 0 = AD0DIO0
bit 1 = AD1DIO1
bit 2 = AD2DIO2
bit 3 = AD3DIO3
bit 4 = AD4DIO4
```

(continues on next page)

(continued from previous page)

```
bit 5 = AD5DIO5ASSOC
bit 6 = NA
bit 7 = Supply Voltage Value

Example: mask of 0x03 means AD0, and AD1 enabled.
0 0 0 0 0 0 1 1
```

Class constructor. Instantiates a new IOSample object with the provided parameters.

**Parameters** io\_sample\_payload (Bytearray) - The payload corresponding to an IO sample.

**Raises** ValueError – If io\_sample\_payload length is less than 5.

# static min\_io\_sample\_payload()

Returns the minimum IO sample payload length.

**Returns** The minimum IO sample payload length.

**Return type** Integer

# digital\_hsb\_mask

Returns the High Significant Byte (HSB) of the digital mask.

**Returns** The HSB of the digital mask.

**Return type** Integer

# digital\_lsb\_mask

Returns the Low Significant Byte (HSB) of the digital mask.

**Returns** The LSB of the digital mask.

**Return type** Integer

# digital\_mask

Returns the combined (HSB + LSB) of the digital mask.

Returns The digital mask.

**Return type** Integer

# digital values

Returns the digital values map.

To verify if this sample contains a valid digital values, use the method <code>IOSample.has digital values()</code>.

**Returns** The digital values map.

Return type Dictionary

# analog\_mask

Returns the analog mask.

Returns the analog mask.

Return type Integer

#### analog\_values

Returns the analog values map.

To verify if this sample contains a valid analog values, use the method <code>IOSample.has\_analog\_values()</code>.

Returns The analog values map.

## **Return type** Dictionary

# power\_supply\_value

Returns the value of the power supply voltage.

To verify if this sample contains the power supply voltage, use the method <code>IOSample.has\_power\_supply\_value()</code>.

#### Returns

The power supply value, *None* if the sample does not contain power supply value.

Return type Integer

#### has\_digital\_values()

Checks whether the IOSample has digital values or not.

**Returns** *True* if the sample has digital values, *False* otherwise.

Return type Boolean

# has\_digital\_value(io\_line)

Returns whether th IO sample contains a digital value for the provided IO line or not.

**Parameters** io\_line (*IOLine*) – The IO line to check if it has a digital value.

Returns

True if the given IO line has a digital value, False otherwise.

Return type Boolean

# has\_analog\_value(io\_line)

Returns whether the given IOLine has an analog value or not.

# Returns

*True* if the given IOLine has an analog value, *False* otherwise.

Return type Boolean

#### has\_analog\_values()

Returns whether the {@code IOSample} has analog values or not.

**Returns** Boolean. *True* if there are analog values, *False* otherwise.

# has\_power\_supply\_value()

Returns whether the IOSample has power supply value or not.

#### Returns

Boolean. True if the given IOLine has a power supply value, False otherwise.

#### get\_digital\_value(io\_line)

Returns the digital value of the provided IO line.

To verify if this sample contains a digital value for the given IOLine, use the method IOSample.  $has\_digital\_value()$ .

**Parameters** io\_line (IOLine) – The IO line to get its digital value.

Returns

**The** *IOValue* **of the given IO line or** *None* **if the IO sample does not contain a digital value for the given IO line.** 

Return type IOValue

## See also:

```
IOLine
IOValue
```

# get\_analog\_value(io\_line)

Returns the analog value of the provided IO line.

To verify if this sample contains an analog value for the given <code>IOLine</code>, use the method <code>IOSample.has\_analog\_value()</code>.

**Parameters** io\_line (IOLine) - The IO line to get its analog value.

# Returns

The analog value of the given IO line or *None* if the IO sample does not contain an analog value for the given IO line.

**Return type** Integer

See also:

*IOLine* 

```
class digi.xbee.io.IOMode
```

Bases: enum. Enum

Enumerates the different Input/Output modes that an IO line can be configured with.

#### DISABLED = 0

Disabled

# SPECIAL\_FUNCTIONALITY = 1

Firmware special functionality

#### PWM = 2

PWM output

# ADC = 2

Analog to Digital Converter

# $DIGITAL_IN = 3$

Digital input

# DIGITAL\_OUT\_LOW = 4

Digital output, Low

#### DIGITAL OUT HIGH = 5

Digital output, High

# 12C\_FUNCTIONALITY = 6

I2C functionality

# digi.xbee.profile module

```
class digi.xbee.profile.FirmwareBaudrate(index, baudrate)
     Bases: enum. Enum
           This class lists the available firmware baudrate options for XBee Profiles.
           Inherited properties:
                name (String): The name of this FirmwareBaudrate.
                value (Integer): The ID of this FirmwareBaudrate.
     Values:
           FirmwareBaudrate.BD 1200 = (0, 1200)
          FirmwareBaudrate.BD_2400 = (1, 2400)
          FirmwareBaudrate.BD_4800 = (2, 4800)
           FirmwareBaudrate.BD 9600 = (3, 9600)
          FirmwareBaudrate.BD 19200 = (4, 19200)
          FirmwareBaudrate.BD_38400 = (5, 38400)
          FirmwareBaudrate.BD_57600 = (6, 57600)
          FirmwareBaudrate.BD_115200 = (7, 115200)
          FirmwareBaudrate.BD 230400 = (8, 230400)
          FirmwareBaudrate.BD_460800 = (9, 460800)
          FirmwareBaudrate.BD_921600 = (10, 921600)
     index
           Returns the index of the FirmwareBaudrate element.
                 Returns Index of the FirmwareBaudrate element.
                Return type Integer
     baudrate
           Returns the baudrate of the FirmwareBaudrate element.
                Returns Baudrate of the FirmwareBaudrate element.
                Return type Integer
class digi.xbee.profile.FirmwareParity(index, parity)
     Bases: enum. Enum
           This class lists the available firmware parity options for XBee Profiles.
           Inherited properties:
                name (String): The name of this FirmwareParity.
                value (Integer): The ID of this FirmwareParity.
     Values:
           FirmwareParity.NONE = (0, <sphinx.ext.autodoc.importer._MockObject object at 0x7f02d6d0c5d0>)
           FirmwareParity.EVEN = (1, <sphinx.ext.autodoc.importer._MockObject object at 0x7f02d6a48310>)
```

**FirmwareParity.ODD** = (2, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7f02d6e04850>) **FirmwareParity.MARK** = (3, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7f02d5e20790>) **FirmwareParity.SPACE** = (4, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7f02d5e20b10>)

#### index

Returns the index of the FirmwareParity element.

**Returns** Index of the *FirmwareParity* element.

Return type Integer

# parity

Returns the parity of the *FirmwareParity* element.

**Returns** Parity of the *FirmwareParity* element.

Return type String

```
class digi.xbee.profile.FirmwareStopbits(index, stop_bits)
```

Bases: enum. Enum

This class lists the available firmware stop bits options for XBee Profiles.

# Inherited properties:

**name** (String): The name of this *FirmwareStopbits*. **value** (Integer): The ID of this *FirmwareStopbits*.

#### Values:

**FirmwareStopbits.SB\_1** = (0, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7f02d6013d50>) **FirmwareStopbits.SB\_2** = (1, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7f02d6013390>) **FirmwareStopbits.SB\_1\_5** = (2, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7f02d6111650>)

# index

Returns the index of the FirmwareStopbits element.

**Returns** Index of the *FirmwareStopbits* element.

Return type Integer

### stop\_bits

Returns the stop bits of the *FirmwareStopbits* element.

**Returns** Stop bits of the *FirmwareStopbits* element.

Return type Float

# class digi.xbee.profile.FlashFirmwareOption(code, description)

Bases: enum. Enum

This class lists the available flash firmware options for XBee Profiles.

Inherited properties:

```
value (Integer): The ID of this FlashFirmwareOption.
     Values:
           FlashFirmwareOption.FLASH_ALWAYS = (0, 'Flash always')
           FlashFirmwareOption.FLASH DIFFERENT = (1, 'Flash firmware if it is different')
           FlashFirmwareOption.DONT_FLASH = (2, 'Do not flash firmware')
     code
           Returns the code of the FlashFirmwareOption element.
                Returns Code of the FlashFirmwareOption element.
                Return type Integer
     description
           Returns the description of the FlashFirmwareOption element.
                Returns Description of the FlashFirmwareOption element.
                Return type String
class digi.xbee.profile.XBeeSettingType (tag, description)
     Bases: enum. Enum
           This class lists the available firmware setting types.
           Inherited properties:
                name (String): The name of this XBeeSettingType.
                value (Integer): The ID of this XBeeSettingType.
     Values:
           XBeeSettingType.NUMBER = ('number', 'Number')
           XBeeSettingType.COMBO = ('combo', 'Combo')
           XBeeSettingType.TEXT = ('text', 'Text')
           XBeeSettingType.BUTTON = ('button', 'Button')
           XBeeSettingType.NO_TYPE = ('none', 'No type')
     tag
           Returns the tag of the XBeeSettingType element.
                Returns Tag of the XBeeSettingType element.
                Return type String
     description
```

Returns the description of the *XBeeSettingType* element.

Return type String

**Returns** Description of the *XBeeSettingType* element.

**name** (String): The name of this *FlashFirmwareOption*.

```
class digi.xbee.profile.XBeeSettingFormat(tag, description)
     Bases: enum. Enum
           This class lists the available text firmware setting formats.
           Inherited properties:
                name (String): The name of this XBeeSettingFormat.
                value (Integer): The ID of this XBeeSettingFormat.
     Values:
           XBeeSettingFormat.HEX = ('HEX', 'Hexadecimal')
           XBeeSettingFormat.ASCII = ('ASCII', 'ASCII')
           XBeeSettingFormat.IPV4 = ('IPV4', 'IPv4')
           XBeeSettingFormat.IPV6 = ('IPV6', 'IPv6')
           XBeeSettingFormat.PHONE = ('PHONE', 'phone')
           XBeeSettingFormat.NO_FORMAT = ('none', 'No format')
     tag
           Returns the tag of the XBeeSettingFormat element.
                Returns Tag of the XBeeSettingFormat element.
                Return type String
     description
           Returns the description of the XBeeSettingFormat element.
                Returns Description of the XBeeSettingFormat element.
                Return type String
class digi.xbee.profile.XBeeProfileSetting(name, setting_type, setting_format, value)
     Bases: object
     This class represents an XBee profile setting and provides information like the setting name, type, format and
     value.
     Class constructor. Instantiates a new XBeeProfileSetting with the given parameters.
           Parameters
                   • name (String) - Setting name.
                   • setting_type (XBeeSettingType) - Setting type.
                   • setting_format (XBeeSettingType) – Setting format.
                   • value (String) – Setting value.
     name
           Returns the XBee setting name.
                Returns XBee setting name.
                Return type String
     type
```

Returns the XBee setting type.

**Returns** XBee setting type.

### Return type XBeeSettingType

#### format

Returns the XBee setting format.

**Returns** XBee setting format.

**Return type** XBeeSettingFormat

#### value

Returns the XBee setting value as string.

**Returns** XBee setting value as string.

**Return type** String

### bytearray\_value

Returns the XBee setting value as bytearray to be set in the device.

**Returns** XBee setting value as bytearray to be set in the device.

**Return type** Bytearray

### exception digi.xbee.profile.ReadProfileException

Bases: digi.xbee.exception.XBeeException

This exception will be thrown when any problem reading the XBee profile occurs.

All functionality of this class is the inherited from Exception.

#### with traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

### exception digi.xbee.profile.UpdateProfileException

Bases: digi.xbee.exception.XBeeException

This exception will be thrown when any problem updating the XBee profile into a device occurs.

All functionality of this class is the inherited from Exception.

#### with\_traceback()

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

#### class digi.xbee.profile.XBeeProfile(profile\_file)

Bases: object

Helper class used to manage serial port break line in a parallel thread.

Class constructor. Instantiates a new XBeeProfile with the given parameters.

**Parameters** profile\_file (String) - Path of the '.xpro' profile file.

### Raises

- ProfileReadException If there is any error reading the profile file.
- ValueError If the provided profile file is not valid

### open()

Opens the profile so its components are accessible from properties firmware\_description\_file, file\_system\_path, remote\_file\_system\_image, and bootloader\_file.

The user is responsible for closing the profile when done with it.

Raises ProfileReadException - If there is any error opening the profile.

### See also:

```
close()
     is_open()
close()
     Closes the profile. Its components are no more accessible.
     See also:
     open()
     is_open()
is_open()
     Returns True if the profile is opened, False otherwise.
     See also:
     open()
     close()
get_setting_default_value (setting_name)
     Returns the default value of the given firmware setting.
           Parameters setting_name (String or ATStringCommand) - Name of the setting to
                retrieve its default value.
           Returns
                Default value of the setting, None if the setting is not found or it has no default
                    value.
           Return type String
profile_file
     Returns the profile file.
           Returns Profile file.
           Return type String
version
     Returns the profile version.
           Returns Profile version.
           Return type String
flash_firmware_option
     Returns the profile flash firmware option.
           Returns Profile flash firmware option.
           Return type FlashFirmwareOption
     See also:
```

#### FlashFirmwareOption

# description

Returns the profile description.

Returns Profile description.

Return type String

### reset\_settings

Returns whether the settings of the XBee will be reset before applying the profile ones or not.

#### Returns

True if the settings of the XBee will be reset before applying the profile ones, False otherwise

Return type Boolean

### has\_local\_filesystem

Returns whether the profile has local filesystem information or not.

#### Returns

*True* if the profile has local filesystem information, *False* otherwise.

Return type Boolean

### has\_remote\_filesystem

Returns whether the profile has remote filesystem information or not.

### Returns

*True* if the profile has remote filesystem information, *False* otherwise.

Return type Boolean

### has\_filesystem

Returns whether the profile has filesystem information (local or remote) or not.

### Returns

*True* if the profile has filesystem information (local or remote), *False* otherwise.

**Return type** Boolean

### has\_local\_firmware\_files

Returns whether the profile has local firmware binaries.

#### Returns

*True* if the profile has local firmware files, *False* otherwise.

Return type Boolean

### has\_remote\_firmware\_files

Returns whether the profile has remote firmware binaries.

#### Returns

*True* if the profile has remote firmware files, *False* otherwise.

Return type Boolean

### has firmware files

Returns whether the profile has firmware binaries (local or remote).

#### Returns

True if the profile has local or remote firmware files, False otherwise.

Return type Boolean

### profile\_settings

Returns all the firmware settings that the profile configures.

#### Returns

## List with all the firmware settings that the profile configures

(XBeeProfileSetting).

Return type Dict

#### firmware\_version

Returns the compatible firmware version of the profile.

**Returns** Compatible firmware version of the profile.

**Return type** Integer

#### hardware\_version

Returns the compatible hardware version of the profile.

**Returns** Compatible hardware version of the profile.

Return type Integer

### compatibility\_number

Returns the compatibility number of the profile.

**Returns** The compatibility number, *None* if not defined.

Return type Integer

### region\_lock

Returns the region lock of the profile.

**Returns** The region lock, *None* if not defined.

Return type Integer

### profile\_description\_file

Returns the path of the profile description file.

**Returns** Path of the profile description file.

Return type String

#### firmware description file

Returns the path of the profile firmware description file.

**Returns** Path of the profile firmware description file.

Return type String

### file\_system\_path

Returns the profile file system path. *None* until the profile is extracted.

**Returns** Path of the profile file system directory.

Return type String

### remote\_file\_system\_image

Returns the path of the remote OTA file system image. *None* until the profile is extracted.

**Returns** Path of the remote OTA file system image.

**Return type** String

### bootloader\_file

Returns the profile bootloader file path. *None* until the profile is extracted.

**Returns** Path of the profile bootloader file.

Return type String

### protocol

Returns the profile XBee protocol.

Returns Profile XBee protocol.

Return type XBeeProtocol

```
digi.xbee.profile.apply_xbee_profile(target, profile_path, timeout=None, progress_callback=None)
```

Applies the given XBee profile into the given XBee. If a serial port is provided as *target*, the XBee profile must include the firmware binaries, that are always programmed. In this case, a restore defaults is also performed before applying settings in the profile (no matter if the profile is configured to do so or not). If the value of 'AP' (operating mode) in the profile is not an API mode or it is not defined, XBee is configured to use API 1.

#### **Parameters**

- **target** (String or *AbstractXBeeDevice*) Target to apply profile to. String: serial port identifier. *AbstractXBeeDevice*: XBee to apply the profile.
- **profile\_path** (*String*) path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the apply profile.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

#### Raises

- ValueError If the XBee profile or the XBee device is not valid.
- *UpdateProfileException* If there is any error during the update XBee profile operation.

### digi.xbee.reader module

```
class digi.xbee.reader.XBeeEvent
    Bases: list
```

This class represents a generic XBee event.

New event callbacks can be added here following this prototype:

```
def callback_prototype(*args, **kwargs):
    #do something...
```

All of them will be executed when the event is fired.

See also:

```
list (Python standard class)
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.PacketReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives any packet, independent of its frame type.
     The callbacks for handle this events will receive the following arguments:
              1. received_packet (XBeeAPIPacket): Received packet.
     See also:
     XBeeAPIPacket
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
```

```
copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.PacketReceivedFrom
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives any packet, independent of its frame type.
     The callbacks for handle this events will receive the following arguments:
              1. received_packet (XBeeAPIPacket): Received packet.
              2. sender (RemoteXBeeDevice): Remote XBee who sent the packet.
     See also:
     RemoteXBeeDevice
     XBeeAPIPacket
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
```

```
extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.DataReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives data.
     The callbacks for handle this events will receive the following arguments:
              1. message (XBeeMessage): Message containing the data received, the sender and the time.
     See also:
     XBeeEvent
     XBeeMessage
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
```

```
pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.ModemStatusReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when a XBee receives a modem status packet.
     The callbacks for handle this events will receive the following arguments:
              1. modem_status (ModemStatus): Modem status received.
     See also:
     XBeeEvent
     ModemStatus
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
```

```
reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.IOSampleReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when a XBee receives an IO packet.
     This includes:
        1. IO data sample RX indicator packet.
        2. RX IO 16 packet.
        3. RX IO 64 packet.
     The callbacks that handle this event will receive the following arguments:
              1. io_sample (IOSample): Received IO sample.
              2. sender (RemoteXBeeDevice): Remote XBee who sent the packet.
              3. time (Integer): the time in which the packet was received.
     See also:
      IOSample
     RemoteXBeeDevice
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
```

Raises ValueError if the value is not present.

```
reverse()
          Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.NetworkModified
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when the network is being modified by the addition of a new node, an existing node informa-
     tion is updated, a node removal, or when the network items are cleared.
     The callbacks that handle this event will receive the following arguments:
             1. event_type (digi.xbee.devices.NetworkEventType): Network event type.
             2. reason (digi.xbee.devices.NetworkEventReason): Reason of the event.
                                                                               digi.xbee.devices.
                          (digi.xbee.devices.XBeeDevice
                                                                      or
                RemoteXBeeDevice): Node added, updated or removed from the network.
     See also:
     digi.xbee.devices.NetworkEventReason
     digi.xbee.devices.NetworkEventType
     digi.xbee.devices.RemoteXBeeDevice
     digi.xbee.devices.XBeeDevice
     XBeeEvent
     append()
          Append object to the end of the list.
     clear()
          Remove all items from list.
     copy()
          Return a shallow copy of the list.
     count()
          Return number of occurrences of value.
     extend()
          Extend list by appending elements from the iterable.
     index()
          Return first index of value.
          Raises ValueError if the value is not present.
     insert()
          Insert object before index.
     pop()
          Remove and return item at index (default last).
          Raises IndexError if list is empty or index is out of range.
     remove()
          Remove first occurrence of value.
```

Raises ValueError if the value is not present.

```
reverse()
          Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.DeviceDiscovered
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee discovers another remote XBee during a discovering operation.
     The callbacks that handle this event will receive the following arguments:
              1. discovered_device (RemoteXBeeDevice): Discovered remote XBee.
     See also:
     RemoteXBeeDevice
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
          Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
          Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
          Remove first occurrence of value.
          Raises ValueError if the value is not present.
     reverse()
          Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.DiscoveryProcessFinished
     Bases: digi.xbee.reader.XBeeEvent
```

This event is fired when the discovery process finishes, either successfully or due to an error.

### The callbacks that handle this event will receive the following arguments:

```
1. status (NetworkDiscoveryStatus): Network discovery status.
```

2. description (String, optional): Description of the discovery status.

See also:

```
NetworkDiscoveryStatus
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.ExplicitDataReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives an explicit data packet.
```

The callbacks for handle this events will receive the following arguments:

1. message (*ExplicitXBeeMessage*): Message containing the received data, the sender, the time, and explicit data message parameters.

See also:

*XBeeEvent* 

```
XBeeMessage
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.IPDataReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives IP data.
     The callbacks for handle this events will receive the following arguments:
              1. message (IPMessage): Message containing containing the IP address the message belongs to,
                 source and destination ports, IP protocol, and the content (data) of the message.
     See also:
     XBeeEvent
      IPMessage
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
```

```
copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.SMSReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives an SMS.
     The callbacks for handle this events will receive the following arguments:
              1. message (SMSMessage): Message containing the phone number that sent the message and the
                 content (data) of the message.
     See also:
     XBeeEvent
     SMSMessage
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
```

Extend list by appending elements from the iterable.

```
index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.RelayDataReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives a user data relay output packet.
     The callbacks to handle these events will receive the following arguments:
              1. message (UserDataRelayMessage): Message containing the source interface and the content
                 (data) of the message.
     See also:
     XBeeEvent
     UserDataRelayMessage
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
```

```
pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.BluetoothDataReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives data from the Bluetooth interface.
     The callbacks to handle these events will receive the following arguments:
              1. data (Bytearray): Received Bluetooth data.
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
```

```
sort()
           Stable sort IN PLACE.
class digi.xbee.reader.MicroPythonDataReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives data from the MicroPython interface.
     The callbacks to handle these events will receive the following arguments:
              1. data (Bytearray): Received MicroPython data.
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.SocketStateReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives a socket state packet.
     The callbacks to handle these events will receive the following arguments:
              1. socket id (Integer): Socket ID for state reported.
```

2. state (SocketState): Received state.

# *XBeeEvent* append() Append object to the end of the list. clear() Remove all items from list. copy() Return a shallow copy of the list. count() Return number of occurrences of value. extend() Extend list by appending elements from the iterable. index() Return first index of value. Raises ValueError if the value is not present. insert() Insert object before index. pop() Remove and return item at index (default last). Raises IndexError if list is empty or index is out of range. remove() Remove first occurrence of value. Raises ValueError if the value is not present. reverse() Reverse IN PLACE. sort() Stable sort IN PLACE. class digi.xbee.reader.SocketDataReceived Bases: digi.xbee.reader.XBeeEvent This event is fired when an XBee receives a socket receive data packet. The callbacks to handle these events will receive the following arguments: 1. socket\_id (Integer): ID of the socket that received the data. 2. payload (Bytearray): Received data. See also: *XBeeEvent* append()

Append object to the end of the list.

See also:

```
clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.SocketDataReceivedFrom
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when an XBee receives a socket receive from data packet.
     The callbacks to handle these events will receive the following arguments:
              1. socket_id (Integer): ID of the socket that received the data.
              2. address (Tuple): Pair (host, port) of the source address where host is a string representing an
                       IPv4 address like '100.50.200.5', and port is an integer.
              3. payload (Bytearray): Received data.
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
```

```
count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.RouteRecordIndicatorReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when a route record packet is received.
     The callbacks to handle these events will receive the following arguments:
              1. Source (RemoteXBeeDevice): Remote node that sent the route record.
              2. Hops (List): List of intermediate hops 16-bit addresses from closest to source (who sent the
                      route record) to closest to destination (XBee16BitAddress).
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
```

index()

Return first index of value.

```
Raises ValueError if the value is not present.

insert()
    Insert object before index.

pop()
    Remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.

remove()
    Remove first occurrence of value.
    Raises ValueError if the value is not present.

reverse()
    Reverse IN PLACE.

sort()
    Stable sort IN PLACE.

class digi.xbee.reader.RouteInformationReceived
```

Pagage ali ari relaca manadan VD an Franch

Bases: digi.xbee.reader.XBeeEvent

This event is fired when a route information packet is received.

The callbacks to handle these events will receive the following arguments:

- 1. Source event (Integer): Source event (0x11: NACK, 0x12: Trace route)
- 2. **Timestamp (Integer): System timer value on the node generating** this package. The timestamp is in microseconds.
- 3. ACK timeout count (Integer): Number of MAC ACK timeouts that occur.
- 4. **TX blocked count (Integer): Number of times the transmissions was** blocked due to reception in progress.
- 5. Destination address (XBee64BitAddress): 64-bit address of the final destination node.
- 6. Source address (XBee64BitAddress): 64-bit address of the source node.
- 7. **Responder address** (XBee64BitAddress): 64-bit address of of the node that generates this packet after it sends (or attempts to send) the packet to the next hop (successor node)
- Successor address (XBee64BitAddress): 64-bit address of of the next node after the responder in the route towards the destination.

See also:

XBeeEvent

```
append()
Append object to the end of the list.

clear()
Remove all items from list.

copy()
Return a shallow copy of the list.

count()
```

Return number of occurrences of value.

```
extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.RouteReceived
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when a route is received.
     The callbacks to handle these events will receive the following arguments:
              1. source (XBeeDevice): Local node.
              2. destination (RemoteXBeeDevice): Remote node.
              3. hops (List): List of intermediate hops from source node to closest
                                                                                         to
                                                                                                   destination
                      (RemoteXBeeDevice).
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
```

```
insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.InitDiscoveryScan
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when a new network discovery scan is about to start.
     The callbacks to handle these events will receive the following arguments:
              1. Number of scan to start (starting with 1).
              2. Total number of scans.
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
```

```
remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.EndDiscoveryScan
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when a network discovery scan has just finished.
     The callbacks to handle these events will receive the following arguments:
              1. Number of scan that has finished (starting with 1).
              2. Total number of scans.
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
```

Stable sort IN PLACE.

class digi.xbee.reader.FileSystemFrameReceived

```
Bases: digi.xbee.reader.XBeeEvent
     This event is fired when a file system packet is received.
     The callbacks to handle these events will receive the following arguments:
              1. Source (AbstractXBeeDevice): Node that sent the file system frame.
              2. Frame id (Integer): Received frame id.
              3. Command (FSCmd): File system command.
              4. Status (:class: .FSCommandStatus): Status code.
              5. Receive options (Integer): Bitfield indicating receive options. See ReceiveOptions.
     See also:
     XBeeEvent
     append()
           Append object to the end of the list.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
     extend()
           Extend list by appending elements from the iterable.
     index()
           Return first index of value.
           Raises ValueError if the value is not present.
     insert()
           Insert object before index.
     pop()
           Remove and return item at index (default last).
           Raises IndexError if list is empty or index is out of range.
     remove()
           Remove first occurrence of value.
           Raises ValueError if the value is not present.
     reverse()
           Reverse IN PLACE.
     sort()
           Stable sort IN PLACE.
class digi.xbee.reader.PacketListener(comm_iface, xbee_device, queue_max_size=None)
     Bases: threading. Thread
```

This class represents a packet listener, which is a thread that's always listening for incoming packets to the XBee.

When it receives a packet, this class throws an event depending on which packet it is. You can add your own callbacks for this events via certain class methods. This callbacks must have a certain header, see each event documentation.

This class has fields that are events. Its recommended to use only the append() and remove() method on them, or -= and += operators. If you do something more with them, it's for your own risk.

Here are the parameters which will be received by the event callbacks, depending on which event it is in each case:

The following parameters are passed via \*\*kwargs to event callbacks of:

- 1. **PacketReceived:** 1.1 received\_packet (XBeeAPIPacket): Received packet.
- 2. DataReceived
  - 2.1 message (XBeeMessage): Message containing the data received, the sender and the time.
- 3. **ModemStatusReceived** 3.1 modem\_status (ModemStatus): Modem status received.

Class constructor. Instantiates a new PacketListener object with the provided parameters.

#### **Parameters**

- comm\_iface (XBeeCommunicationInterface) Hardware interface to listen to.
- **xbee\_device** (XBeeDevice) XBee that is the listener owner.
- queue\_max\_size (Integer) Maximum size of the XBee queue.

#### daemon

A boolean value indicating whether this thread is a daemon thread.

This must be set before start() is called, otherwise RuntimeError is raised. Its initial value is inherited from the creating thread; the main thread is not a daemon thread and therefore all threads created in the main thread default to daemon = False.

The entire Python program exits when only daemon threads are left.

```
wait_until_started(timeout=None)
```

Blocks until the thread has fully started. If already started, returns immediately.

**Parameters** timeout (Float) – Timeout for the operation in seconds.

### run()

This is the method that will be executing for listening packets.

For each packet, it will execute the proper callbacks.

#### stop()

Stops listening.

#### is\_running()

Returns whether this instance is running or not.

**Returns** *True* if this instance is running, *False* otherwise.

Return type Boolean

#### get\_queue()

Returns the packets queue.

**Returns** Packets queue.

Return type XBeeQueue

#### get\_data\_queue()

Returns the data packets queue.

**Returns** Data packets queue.

Return type XBeeQueue

#### get\_explicit\_queue()

Returns the explicit packets queue.

Returns Explicit packets queue.

Return type XBeeQueue

#### get\_ip\_queue()

Returns the IP packets queue.

**Returns** IP packets queue.

Return type XBeeQueue

### add\_packet\_received\_callback(callback)

Adds a callback for the event PacketReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives one argument.

• The received packet as a XBeeAPIPacket

### add\_packet\_received\_from\_callback(callback)

Adds a callback for the event PacketReceivedFrom.

**Parameters callback** (Function or List of functions) - Callback. Receives two arguments.

- The received packet as a XBeeAPIPacket
- ullet The remote XBee device who has sent the packet as a  ${\it RemoteXBeeDevice}$

### add\_data\_received\_callback(callback)

Adds a callback for the event DataReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives one argument.

• The data received as an XBeeMessage

# $\verb"add_modem_status_received_callback" (callback)$

Adds a callback for the event ModemStatusReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives one argument.

• The modem status as a ModemStatus

# add\_io\_sample\_received\_callback(callback)

Adds a callback for the event IOSampleReceived.

**Parameters callback** (Function or List of functions) – Callback. Receives three arguments.

- The received IO sample as an IOSample
- The remote XBee device who has sent the packet as a RemoteXBeeDevice
- The time in which the packet was received as an Integer

#### add explicit data received callback(callback)

Adds a callback for the event ExplicitDataReceived.

Parameters callback (Function or List of functions) - Callback. Receives one argument.

• The explicit data received as an ExplicitXBeeMessage

### add\_ip\_data\_received\_callback(callback)

Adds a callback for the event IPDataReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives one argument.

• The data received as an IPMessage

#### add\_sms\_received\_callback(callback)

Adds a callback for the event SMSReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives one argument.

• The data received as an SMSMessage

### add\_user\_data\_relay\_received\_callback(callback)

Adds a callback for the event RelayDataReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives one argument.

• The data received as a UserDataRelayMessage

### add\_bluetooth\_data\_received\_callback(callback)

Adds a callback for the event BluetoothDataReceived.

Parameters callback (Function or List of functions) - Callback. Receives one argument.

• The data received as a Bytearray

### add\_micropython\_data\_received\_callback(callback)

Adds a callback for the event MicroPythonDataReceived.

Parameters callback (Function or List of functions) - Callback. Receives one argument.

• The data received as a Bytearray

### add\_socket\_state\_received\_callback(callback)

Adds a callback for the event SocketStateReceived.

Parameters callback (Function or List of functions) - Callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a SocketState

#### add\_socket\_data\_received\_callback (callback)

Adds a callback for the event SocketDataReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives two arguments.

• The socket ID as an Integer.

• The status received as a SocketStatus

#### add socket data received from callback(callback)

Adds a callback for the event SocketDataReceivedFrom.

Parameters callback (Function or List of functions) - Callback. Receives three arguments.

- The socket ID as an Integer.
- A pair (host, port) of the source address where host is a string representing an IPv4 address like '100.50.200.5', and port is an integer.
- The status received as a SocketStatus

### add\_route\_record\_received\_callback (callback)

Adds a callback for the event RouteRecordIndicatorReceived.

**Parameters callback** (Function or List of functions) - Callback. Receives two arguments.

- Source (RemoteXBeeDevice): Remote node that sent the route record.
- Hops (List): List of intermediate hops 16-bit addresses from closest to source (who sent the route record) to closest to destination.

#### add route info received callback (callback)

Adds a callback for the event RouteInformationReceived.

**Parameters callback** (Function or List of functions) – Callback. Receives eight arguments.

- Source event (Integer): Source event (0x11: NACK, 0x12: Trace route)
- Timestamp (Integer): System timer value on the node generating this package. The timestamp is in microseconds.
- ACK timeout count (Integer): Number of MAC ACK timeouts that occur.
- TX blocked count (Integer): Number of times the transmissions was blocked due to reception in progress.
- Destination address (XBee64BitAddress): 64-bit address of the final destination node.
- Source address (XBee64BitAddress): 64-bit address of the source node.
- Responder address (XBee64BitAddress): 64-bit address of the node that generated this packet after it sent (or attempted to send) the packet to the next hop (successor node)
- Successor address (XBee64BitAddress): 64-bit address of the next node after the responder in the route towards the destination.

### $\verb"add_fs_frame_received_callback" (callback)$

Adds a callback for the event FileSystemFrameReceived.

**Parameters callback** (Function or List of functions) – Callback. Receives four arguments.

- Source (AbstractXBeeDevice): Node that sent the file system frame.
- Frame id (Integer): Received frame id.
- Command (FSCmd): File system command.

• Receive options (Integer): Bitfield indicating receive options. See ReceiveOptions.

### del\_packet\_received\_callback (callback)

Deletes a callback for the callback list of PacketReceived event.

Parameters callback (Function) - Callback to delete.

**Raises** ValueError – If callback is not in the callback list of PacketReceived event.

### del\_packet\_received\_from\_callback (callback)

Deletes a callback for the callback list of PacketReceivedFrom event.

Parameters callback (Function) - Callback to delete.

Raises ValueError – If callback is not in the callback list of PacketReceivedFrom event.

### del\_data\_received\_callback(callback)

Deletes a callback for the callback list of DataReceived event.

**Parameters** callback (Function) – Callback to delete.

Raises ValueError - If callback is not in the callback list of DataReceived event.

### del\_modem\_status\_received\_callback(callback)

Deletes a callback for the callback list of ModemStatusReceived event.

**Parameters** callback (Function) – Callback to delete.

Raises ValueError – If *callback* is not in the callback list of *ModemStatusReceived* event.

### del\_io\_sample\_received\_callback (callback)

Deletes a callback for the callback list of IOSampleReceived event.

Parameters callback (Function) - Callback to delete.

Raises ValueError — If *callback* is not in the callback list of *IOSampleReceived* event.

### del\_explicit\_data\_received\_callback(callback)

Deletes a callback for the callback list of *ExplicitDataReceived* event.

**Parameters** callback (Function) – Callback to delete.

Raises ValueError — If callback is not in the callback list of ExplicitDataReceived event.

#### del ip data received callback(callback)

Deletes a callback for the callback list of IPDataReceived event.

Parameters callback (Function) - Callback to delete.

Raises ValueError - If callback is not in the callback list of IPDataReceived event.

### del\_sms\_received\_callback(callback)

Deletes a callback for the callback list of SMSReceived event.

**Parameters** callback (Function) – Callback to delete.

Raises ValueError - If callback is not in the callback list of SMSReceived event.

### del\_user\_data\_relay\_received\_callback (callback)

Deletes a callback for the callback list of RelayDataReceived event.

Parameters callback (Function) - Callback to delete.

Raises ValueError - If callback is not in the callback list of RelayDataReceived event.

# ${\tt del\_bluetooth\_data\_received\_callback}\ (callback)$

Deletes a callback for the callback list of BluetoothDataReceived event.

Parameters callback (Function) - Callback to delete.

Raises ValueError - If callback is not in the callback list of BluetoothDataReceived event.

### del\_micropython\_data\_received\_callback(callback)

Deletes a callback for the callback list of MicroPythonDataReceived event.

**Parameters** callback (Function) – Callback to delete.

**Raises** ValueError — If *callback* is not in the *callback* list of *MicroPythonDataReceived* event.

### ${\tt del\_socket\_state\_received\_callback}\ (callback)$

Deletes a callback for the callback list of SocketStateReceived event.

**Parameters** callback (Function) – Callback to delete.

**Raises** ValueError – If *callback* is not in the callback list of *SocketStateReceived* event.

### del\_socket\_data\_received\_callback(callback)

Deletes a callback for the callback list of SocketDataReceived event.

**Parameters** callback (Function) – Callback to delete.

Raises ValueError – If callback is not in the callback list of SocketDataReceived event.

### del\_socket\_data\_received\_from\_callback(callback)

Deletes a callback for the callback list of SocketDataReceivedFrom event.

Parameters callback (Function) - Callback to delete.

Raises ValueError - If callback is not in the callback list of SocketDataReceivedFrom event.

### del\_route\_record\_received\_callback (callback)

Deletes a callback for the callback list of RouteRecordIndicatorReceived event.

Parameters callback (Function) - Callback to delete.

Raises ValueError - If callback is not in the callback list of RouteRecordIndicatorReceived event.

#### del\_route\_info\_callback(callback)

Deletes a callback for the callback list of RouteInformationReceived event.

**Parameters** callback (Function) – Callback to delete.

Raises ValueError - If callback is not in the callback list of RouteInformationReceived event.

#### del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

Parameters callback (Function) - Callback to delete.

**Raises** ValueError - If *callback* is not in the callback list of *FileSystemFrameReceived* event.

### get\_packet\_received\_callbacks()

Returns the list of registered callbacks for received packets.

**Returns** List of *PacketReceived* events.

Return type List

#### get\_packet\_received\_from\_callbacks()

Returns the list of registered callbacks for received packets.

**Returns** List of *PacketReceivedFrom* events.

**Return type** List

#### get\_data\_received\_callbacks()

Returns the list of registered callbacks for received data.

**Returns** List of *DataReceived* events.

**Return type** List

#### get\_modem\_status\_received\_callbacks()

Returns the list of registered callbacks for received modem status.

**Returns** List of ModemStatusReceived events.

**Return type** List

#### get\_io\_sample\_received\_callbacks()

Returns the list of registered callbacks for received IO samples.

**Returns** List of IOSampleReceived events.

**Return type** List

### get\_explicit\_data\_received\_callbacks()

Returns the list of registered callbacks for received explicit data.

**Returns** List of ExplicitDataReceived events.

**Return type** List

### get\_ip\_data\_received\_callbacks()

Returns the list of registered callbacks for received IP data.

Returns List of IPDataReceived events.

Return type List

### get\_sms\_received\_callbacks()

Returns the list of registered callbacks for received SMS.

**Returns** List of SMSReceived events.

Return type List

### get\_user\_data\_relay\_received\_callbacks()

Returns the list of registered callbacks for received user data relay.

**Returns** List of RelayDataReceived events.

Return type List

### get\_bluetooth\_data\_received\_callbacks()

Returns the list of registered callbacks for received Bluetooth data.

**Returns** List of BluetoothDataReceived events.

#### **Return type** List

#### get\_micropython\_data\_received\_callbacks()

Returns the list of registered callbacks for received MicroPython data.

Returns List of MicroPythonDataReceived events.

**Return type** List

#### get\_socket\_state\_received\_callbacks()

Returns the list of registered callbacks for received socket state.

Returns List of SocketStateReceived events.

**Return type** List

### get\_socket\_data\_received\_callbacks()

Returns the list of registered callbacks for received socket data.

Returns List of SocketDataReceived events.

**Return type** List

### get\_socket\_data\_received\_from\_callbacks()

Returns the list of registered callbacks for received socket data from.

**Returns** List of SocketDataReceivedFrom events.

**Return type** List

### get\_route\_record\_received\_callbacks()

Returns the list of registered callbacks for received route records.

Returns List of RouteRecordIndicatorReceived events.

Return type List

## get\_route\_info\_callbacks()

Returns the list of registered callbacks for received route information packets.

Returns List of RouteInformationReceived events.

**Return type** List

### get\_fs\_frame\_received\_callbacks()

Returns the list of registered callbacks for received file system packets.

 $\textbf{Returns} \ \ List \ of \ \textit{FileSystemFrameReceived} \ \textbf{events}.$ 

Return type List

#### ident

Thread identifier of this thread or None if it has not been started.

This is a nonzero integer. See the get\_ident() function. Thread identifiers may be recycled when a thread exits and another thread is created. The identifier is available even after the thread has exited.

### isAlive()

Return whether the thread is alive.

This method is deprecated, use is\_alive() instead.

### is\_alive()

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

#### join (timeout=None)

Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call is\_alive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

#### name

A string used for identification purposes only.

It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.

### start()

Start the thread's activity.

It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.

This method will raise a RuntimeError if called more than once on the same thread object.

```
class digi.xbee.reader.XBeeQueue (maxsize=10)
```

Bases: queue.Queue

This class represents an XBee queue.

Class constructor. Instantiates a new XBeeQueue with the provided parameters.

Parameters maxsize (Integer, optional, default=10) - Maximum size of the queue.

```
get (block=True, timeout=None)
```

Returns the first element of the queue if there is some element ready before timeout expires, in case of the timeout is not *None* 

If timeout is *None*, this method is non-blocking. In this case, if there is not any element available, it returns *None*, otherwise it returns an *XBeeAPIPacket*.

### **Parameters**

- block (Boolean) True to block during timeout waiting for a packet, False to not block.
- timeout (Integer, optional) timeout in seconds.

#### Returns

**Packet if there is any packet available** before *timeout* expires. If *timeout* is *None*, the returned value may be *None*.

Return type XBeeAPIPacket

**Raises** TimeoutException – If *timeout* is not *None* and there is not any packet available before the timeout expires.

#### get by remote(remote, timeout=None)

Returns the first element of the queue that had been sent by *remote*, if there is some in the specified timeout.

If timeout is *None*, this method is non-blocking. In this case, if there is not any packet sent by *remote* in the queue, it returns *None*, otherwise it returns an *XBeeAPIPacket*.

#### **Parameters**

- remote (RemoteXBeeDevice) Remote XBee to get its first element from queue.
- timeout (Integer, optional, default=`None`) Timeout in seconds.

#### Returns

**If there is any packet available before** the timeout expires. If timeout is *None*, the returned value may be *None*.

### Return type XBeeAPIPacket

**Raises** TimeoutException – If timeout is not *None* and there is not any packet available that was sent by *remote* before the timeout expires.

### get\_by\_ip (ip\_addr, timeout=None)

Returns the first IP data packet from the queue whose IP address matches the provided address.

If timeout is *None*, this method is non-blocking. In this case, if there is not any packet sent by *ip\_addr* in the queue, it returns *None*, otherwise it returns an *XBeeAPIPacket*.

#### **Parameters**

- **ip\_addr** (ipaddress.IPv4Address) IP address to look for in the list of packets.
- timeout (Integer, optional, default=`None`) Timeout in seconds.

#### Returns

**If there is any packet available before the** timeout expires. If timeout is *None*, the returned value may be *None*.

### Return type XBeeAPIPacket

**Raises** TimeoutException – If timeout is not *None* and there is not any packet available that was sent by  $ip\_addr$  before the timeout expires.

#### empty()

Return True if the queue is empty, False otherwise (not reliable!).

This method is likely to be removed at some point. Use qsize() == 0 as a direct substitute, but be aware that either approach risks a race condition where a queue can grow before the result of empty() or qsize() can be used.

To create code that needs to wait for all queued tasks to be completed, the preferred technique is to use the join() method.

### full()

Return True if the queue is full, False otherwise (not reliable!).

This method is likely to be removed at some point. Use qsize() >= n as a direct substitute, but be aware that either approach risks a race condition where a queue can shrink before the result of full() or qsize() can be used.

#### get by id(frame id, timeout=None)

Returns the first packet from the queue whose frame ID matches the provided one.

If timeout is *None*, this method is non-blocking. In this case, if there is not any received packet with the provided frame ID in the queue, it returns *None*, otherwise it returns an *XBeeAPIPacket*.

#### **Parameters**

- **frame\_id** (Integer) Frame ID to look for in the list of packets.
- timeout (Integer, optional, default=`None`) Timeout in seconds.

#### Returns

**If there is any packet available before** the timeout expires. If timeout is *None*, the returned value may be *None*.

# Return type XBeeAPIPacket

**Raises** TimeoutException – If timeout is not *None* and there is not any packet available that matches the provided frame ID before the timeout expires.

#### get\_nowait()

Remove and return an item from the queue without blocking.

Only get an item if one is immediately available. Otherwise raise the Empty exception.

#### join()

Blocks until all items in the Queue have been gotten and processed.

The count of unfinished tasks goes up whenever an item is added to the queue. The count goes down whenever a consumer thread calls task\_done() to indicate the item was retrieved and all work on it is complete.

When the count of unfinished tasks drops to zero, join() unblocks.

# put (item, block=True, timeout=None)

Put an item into the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until a free slot is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Full exception if no free slot was available within that time. Otherwise ('block' is false), put an item on the queue if a free slot is immediately available, else raise the Full exception ('timeout' is ignored in that case).

# put\_nowait (item)

Put an item into the queue without blocking.

Only enqueue the item if a free slot is immediately available. Otherwise raise the Full exception.

#### qsize()

Return the approximate size of the queue (not reliable!).

# task\_done()

Indicate that a formerly enqueued task is complete.

Used by Queue consumer threads. For each get() used to fetch a task, a subsequent call to task\_done() tells the queue that the processing on the task is complete.

If a join() is currently blocking, it will resume when all items have been processed (meaning that a task\_done() call was received for every item that had been put() into the queue).

Raises a ValueError if called more times than there were items placed in the queue.

2.6. API reference 1003

```
flush()
```

Clears the queue.

# digi.xbee.recovery module

```
digi.xbee.recovery.recover_device (target)
```

Recovers the XBee from an unknown state and leaves if configured for normal operations.

**Parameters** target (String or XBeeDevice) – Target of the recovery operation.

Raises RecoveryException – If there is any error performing the recovery action.

```
digi.xbee.recovery.enter_at_command_mode(port)
```

Attempts to put this device in AT Command mode.

**Parameters** port – The serial port where the XBee is connected to.

#### Returns

True if the XBee has entered in AT command mode, False otherwise.

Return type Boolean

#### Raises

- SerialTimeoutException If there is any error trying to write to the serial port.
- InvalidOperatingModeException If the XBee is in API mode.

# digi.xbee.sender module

```
class digi.xbee.sender.PacketSender(xbee)
```

Bases: object

Class to send XBee packets.

Class constructor. Instantiates a new Packet Sender object with the provided parameters.

Parameters xbee (XBeeDevice) - The XBee.

```
send_packet (packet)
```

Sends a packet to the XBee. The packet to send is escaped depending on the current operating mode.

**Parameters** packet (XBeePacket) – The packet to send.

#### Raises

- InvalidOperatingModeException If the XBee device's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- XBeeException if the XBee device's communication interface is closed.

# See also:

XBeePacket

# is\_op\_mode\_valid(value)

Returns *True* if the provided value is a valid operating mode for the library.

**Parameters value** (*Bytearray*) – The value to check.

**Returns** *True* for a valid value, *False* otherwise.

Return type Boolean

# at\_response\_received\_cb (response)

Callback to deal with AT command responses and update the corresponding node. Only for internal use.

**Parameters** ((response) – class: .XBeeAPIPacket): The received API packet.

```
class digi.xbee.sender.SyncRequestSender(xbee, packet_to_send, timeout)
    Bases: object
```

Class to synchronously send XBee packets. This means after sending the packet it waits for its response, if the package includes a frame ID, otherwise it does not wait.

Class constructor. Instantiates a new SyncRequestSender object with the provided parameters.

#### **Parameters**

- **xbee** (XBeeDevice) The local XBee to send the packet.
- packet\_to\_send (XBeePacket) The packet to transmit.
- timeout (Integer) Number of seconds to wait. -1 to wait indefinitely.

#### send()

Sends the packet and waits for its corresponding response.

**Returns** Received response packet.

Return type XBeePacket

#### Raises

- InvalidOperatingModeException If the XBee device's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received in the configured timeout.
- XBeeException If the XBee device's communication interface is closed.

#### See also:

XBeePacket

#### xbee

Returns the local XBee to send the packet.

**Returns** Local XBee device.

Return type XBeeDevice

# packet

Returns the packet to send.

Returns Packet to send.

Return type XBeePacket

# timeout

Returns the maximum number of seconds to wait for a response.

**Returns** Timeout to wait for a response.

Return type Integer

2.6. API reference 1005

# digi.xbee.serial module

```
class digi.xbee.serial.FlowControl
    Bases: enum.Enum
```

This class represents all available flow controls.

Bases: sphinx.ext.autodoc.importer.\_MockObject, digi.xbee.comm\_interface. XBeeCommunicationInterface

This class extends the functionality of Serial class (PySerial).

It also introduces a minor change in its behaviour: the serial port is not automatically open when instantiated, only when calling open().

See also:

\_PySerial: https://github.com/pyserial/pyserial

Class constructor. Instantiates a new XBeeSerialPort object with the given port parameters.

#### **Parameters**

- baud\_rate (Integer) Serial port baud rate.
- port (String) Serial port name to use.
- data\_bits (Integer, optional, default=8) Serial data bits.
- stop\_bits (Float, optional, default=1) sSerial stop bits.
- parity (Char, optional, default=`N`) Parity. Default to 'N' (None).
- flow control (Integer, optional, default=`None`) Flow control.
- timeout (Integer, optional, default=0.1) Read timeout (seconds).

# See also:

PySerial: https://github.com/pyserial/pyserial

#### is\_interface\_open

Returns whether the underlying hardware communication interface is active.

**Returns** Boolean. *True* if the interface is active, *False* otherwise.

# write\_frame (frame)

Writes an XBee frame to the underlying hardware interface.

Subclasses may throw specific exceptions to signal implementation specific hardware errors.

**Parameters frame** (*Bytearray*) – The XBee API frame packet to write. If the bytearray does not correctly represent an XBee frame, the behaviour is undefined.

#### read byte()

Synchronous. Reads one byte from serial port.

**Returns** The read byte.

Return type Integer

**Raises** TimeoutException – If there is no bytes ins serial port buffer.

#### read\_bytes (num\_bytes)

Synchronous. Reads the specified number of bytes from the serial port.

**Parameters** num\_bytes (Integer) – the number of bytes to read.

**Returns** the read bytes.

Return type Bytearray

Raises TimeoutException - if the number of bytes read is less than num\_bytes.

# quit\_reading()

Makes the thread (if any) blocking on wait\_for\_frame return.

If a thread was blocked on wait\_for\_frame, this method blocks (for a maximum of 'timeout' seconds) until the blocked thread is resumed.

## wait\_for\_frame (operating\_mode)

Reads the next packet. Starts to read when finds the start delimiter. The last byte read is the checksum.

If there is something in the COM buffer after the start delimiter, this method discards it.

If the method can't read a complete and correct packet, it will return *None*.

**Parameters operating\_mode** (OperatingMode) – The operating mode in which the packet should be read.

# Returns

The read packet as bytearray if a packet is read, *None* otherwise.

**Return type** Bytearray

# read\_existing()

Asynchronous. Reads all bytes in the serial port buffer. May read 0 bytes.

**Returns** The bytes read.

Return type Bytearray

# get\_read\_timeout()

Returns the serial port read timeout.

**Returns** Read timeout in seconds.

Return type Integer

# set\_read\_timeout (read\_timeout)

Sets the serial port read timeout in seconds.

**Parameters** read\_timeout (Integer) - The new serial port read timeout in seconds.

# set\_baudrate (new\_baudrate)

Changes the serial port baudrate.

**Parameters** new\_baudrate (Integer) - The new baudrate to set.

# purge\_port()

Purges the serial port by cleaning the input and output buffers.

2.6. API reference 1007

**apply\_profile** (*xbee*, *profile\_path*, *timeout=None*, *progress\_callback=None*) Applies the given XBee profile to the XBee device.

#### **Parameters**

- xbee (AbstractXBeeDevice) Local or remote XBee node to be updated.
- **profile\_path** (*String*) Path of the XBee profile file to apply.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the apply profile.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Receives two arguments:
  - The current apply profile task as a String
  - The current apply profile task percentage as an Integer

# Raises

- XBeeException If the local XBee is not open.
- InvalidOperatingModeException If the local XBee operating mode is invalid.
- UpdateProfileException If there is any error applying the XBee profile.
- OperationNotSupportedException If XBee profiles are not supported in the XBee.

#### close()

Terminates the underlying hardware communication interface.

Subclasses may throw specific exceptions to signal implementation specific hardware errors.

# get\_local\_xbee\_info()

Returns a tuple with the local XBee information.

This is used when opening the local XBee. If this information is provided, it is used as internal XBee data, if not provided, the data is requested to the XBee.

#### Returns

**Tuple with local XBee information: operation mode (int),** hardware version (int), firmware version (int), 64-bit address (string), 16-bit address (string), node identifier (string), and role (int).

# Return type Tuple

#### get network(local xbee)

Returns the XBeeNetwork object associated to the XBeeDevice associated to this XBeeCommunication-Interface.

Some XBeeCommunicationInterface implementations may need to handle the 'XBeeNetwork associated to the XBeeDevice themselves. If that is the case, a implementation-specific XBeeNetwork object that complains to the generic XBeeNetwork class will be returned. Otherwise, this method returns None and the associated XBeeNetwork is handled as for a serial-connected XBeeDevice.

**Parameters** local\_xbee (XBeeDevice) - The local XBee device.

#### Returns

**class:** .XBeeNetwork: None if the XBeeNetwork should handled as usual, otherwise a XBeeNetwork object.

# open()

Establishes the underlying hardware communication interface.

Subclasses may throw specific exceptions to signal implementation specific errors.

#### supports\_apply\_profile()

Returns if the interface supports the apply profile feature.

**Returns** *True* if it is supported, *False* otherwise.

Return type Boolean

#### supports\_update\_firmware()

Returns if the interface supports the firmware update feature.

**Returns** *True* if it is supported, *False* otherwise.

Return type Boolean

#### timeout

Returns the read timeout.

**Returns** Read timeout in seconds.

**Return type** Integer

update\_firmware (xbee, xml\_fw\_file, xbee\_fw\_file=None, bootloader\_fw\_file=None, timeout=None, progress\_callback=None)

Performs a firmware update operation of the provided XBee.

#### **Parameters**

- **xbee** (AbstractXBeeDevice) Local or remote XBee node to be updated.
- **xml\_fw\_file** (*String*) Path of the XML file that describes the firmware to upload.
- **xbee\_fw\_file** (String, optional) Location of the XBee binary firmware file.
- bootloader\_fw\_file (String, optional) Location of the bootloader binary firmware file.
- **timeout** (*Integer*, *optional*) Maximum time to wait for target read operations during the update process.
- **progress\_callback** (Function, optional) Function to execute to receive progress information. Receives two arguments:
  - The current update task as a String
  - The current update task percentage as an Integer

# Raises

- XBeeException If the local XBee is not open.
- InvalidOperatingModeException If the local XBee operating mode is invalid.
- OperationNotSupportedException If the firmware update is not supported in the XBee.
- FirmwareUpdateException If there is any error performing the firmware update.

2.6. API reference 1009

# digi.xbee.xsocket module

This class represents an XBee socket and provides methods to create, connect, bind and close a socket, as well as send and receive data with it.

Class constructor. Instantiates a new XBee socket object for the given XBee device.

#### **Parameters**

- **xbee\_device** (XBeeDevice) XBee device of the socket.
- **ip\_protocol** (*IPProtocol*) protocol of the socket.

#### Raises

- ValueError if xbee\_device is None or if xbee\_device is not an instance of CellularDevice.
- ValueError if *ip\_protocol* is *None*.
- XBeeException if the connection with the XBee device is not open.

#### connect (address)

Connects to a remote socket at the given address.

**Parameters address** (*Tuple*) – A pair (*host, port*) where *host* is the domain name or string representation of an IPv4 and *port* is the numeric port value.

#### **Raises**

- TimeoutException If the connect response is not received in the configured timeout.
- ValueError If *address* is *None* or not a pair (*host*, *port*).
- ValueError If *port* is less than 1 or greater than 65535.
- XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the connect status is not SUCCESS.

#### bind(address)

Binds the socket to the given address. The socket must not already be bound.

**Parameters address** (*Tuple*) – A pair (*host, port*) where *host* is the local interface (not used) and *port* is the numeric port value.

#### Raises

- TimeoutException If the bind response is not received in the configured timeout.
- ValueError If *address* is *None* or not a pair (*host, port*).
- ValueError If *port* is less than 1 or greater than 65535.
- XBeeException If the connection with the XBee device is not open.
- $\bullet$  XBeeSocketException If the bind status is not SUCCESS.
- XBeeSocketException If the socket is already bound.

# listen(backlog=1)

Enables a server to accept connections.

**Parameters backlog** (Integer, optional) – The number of unaccepted connections that the system will allow before refusing new connections. If specified, it must be at least 0 (if it is lower, it is set to 0).

Raises XBeeSocketException - If the socket is not bound.

#### accept()

Accepts a connection. The socket must be bound to an address and listening for connections.

#### Returns

A pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is a pair (host, port) with the address bound to the socket on the other end of the connection.

# Return type Tuple

#### Raises

- XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the socket is not bound or not listening.

#### gettimeout()

Returns the configured socket timeout in seconds.

**Returns** The configured timeout in seconds.

Return type Integer

#### settimeout (timeout)

Sets the socket timeout in seconds.

**Parameters** timeout (Integer) – The new socket timeout in seconds.

# getblocking()

Returns whether the socket is in blocking mode or not.

**Returns** *True* if the socket is in blocking mode, *False* otherwise.

Return type Boolean

#### setblocking(flag)

Sets the socket in blocking or non-blocking mode.

**Parameters flag** (Boolean) – *True* to set the socket in blocking mode, *False* to set it in no blocking mode and configure the timeout with the default value (5 seconds).

#### recv (bufsize)

Receives data from the socket.

**Parameters** bufsize (Integer) – The maximum amount of data to be received at once.

**Returns** The data received.

Return type Bytearray

**Raises** ValueError – If *bufsize* is less than 1.

#### recvfrom (bufsize)

Receives data from the socket.

**Parameters** bufsize (Integer) – The maximum amount of data to be received at once.

Returns

2.6. API reference 1011

**Pair containing the data received** (Bytearray) and the address of the socket sending the data. The address is also a pair (*host*, *port*) where *host* is the string representation of an IPv4 and *port* is the numeric port value.

**Return type** Tuple (Bytearray, Tuple)

**Raises** ValueError – If *bufsize* is less than 1.

#### send (data)

Sends data to the socket and returns the number of bytes sent. The socket must be connected to a remote socket. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data.

**Parameters data** (Bytearray) – The data to send.

**Returns** The number of bytes sent.

Return type Integer

#### Raises

- ValueError If the data to send is None.
- ValueError If the number of bytes to send is  $\theta$ .
- XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the socket is not valid.
- XBeeSocketException If the socket is not open.

#### sendall(data)

Sends data to the socket. The socket must be connected to a remote socket. Unlike *send()*, this method continues to send data from bytes until either all data has been sent or an error occurs. *None* is returned on success. On error, an exception is raised, and there is no way to determine how much data, if any, was successfully sent.

**Parameters data** (*Bytearray*) – The data to send.

#### Raises

- TimeoutException If the send status response is not received in the configured timeout.
- ValueError If the data to send is *None*.
- ValueError If the number of bytes to send is  $\theta$ .
- $\bullet$  XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the socket is not valid.
- XBeeSocketException If the send status is not SUCCESS.
- XBeeSocketException If the socket is not open.

#### sendto (data, address)

Sends data to the socket. The socket should not be connected to a remote socket, since the destination socket is specified by *address*.

#### **Parameters**

- data (Bytearray) The data to send.
- address (Tuple) The address of the destination socket. It must be a pair (host, port) where host is the domain name or string representation of an IPv4 and port is the numeric port value.

**Returns** The number of bytes sent.

#### Return type Integer

#### Raises

- TimeoutException If the send status response is not received in the configured timeout.
- ValueError If the data to send is None.
- ValueError If the number of bytes to send is  $\theta$ .
- XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the socket is already open.
- XBeeSocketException If the send status is not SUCCESS.

## close()

Closes the socket.

#### Raises

- TimeoutException If the close response is not received in the configured timeout.
- XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the close status is not SUCCESS.

# setsocketopt (option, value)

Sets the value of the given socket option.

# **Parameters**

- **option** (*SocketOption*) The socket option to set its value.
- **value** (*Bytearray*) The new value of the socket option.

#### Raises

- TimeoutException If the socket option response is not received in the configured timeout.
- ValueError If the option to set is *None*.
- ValueError If the value of the option is *None*.
- XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the socket option response status is not SUC-CESS.

# getsocketopt(option)

Returns the value of the given socket option.

**Parameters** option (SocketOption) – The socket option to get its value.

**Returns** The value of the socket option.

**Return type** Bytearray

# Raises

- TimeoutException If the socket option response is not received in the configured timeout.
- ValueError If the option to set is *None*.

2.6. API reference 1013

- XBeeException If the connection with the XBee device is not open.
- XBeeSocketException If the socket option response status is not SUC-CESS.

# add\_socket\_state\_callback(callback)

Adds a callback for the event digi.xbee.reader.SocketStateReceived.

**Parameters** callback (Function) – The callback. Receives two arguments.

- The socket ID as an Integer.
- The state received as a SocketState

# del\_socket\_state\_callback(callback)

Deletes a callback for the callback list of digi.xbee.reader.SocketStateReceived event.

**Parameters** callback (Function) – The callback to delete.

Raises ValueError - If callback is not in the callback list of digi.xbee.reader. SocketStateReceived event.

#### get sock info()

Returns the information of this socket.

**Returns** The socket information.

Return type SocketInfo

#### Raises

- InvalidOperatingModeException If the XBee device's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- TimeoutException If the response is not received before the read timeout expires.
- XBeeException If the XBee device's communication interface is closed.

#### See also:

SocketInfo

# is connected

Returns whether the socket is connected or not.

**Returns** *True* if the socket is connected *False* otherwise.

Return type Boolean

# $\mathsf{CHAPTER}\,3$

# Indices and tables

- genindex
- modindex
- search

$\sim$ L			 $\Box$	
CH	Α	РΊ	К	4

License

Copyright 2017-2021, Digi International Inc.

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, you can obtain one at http://mozilla.org/MPL/2.0/.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Digi International Inc. 11001 Bren Road East, Minnetonka, MN 55343

1018 Chapter 4. License

# Python Module Index

```
d
                                           digi.xbee.recovery, 1004
                                           digi.xbee.sender, 1004
digi, 124
                                           digi.xbee.serial, 1006
digi.xbee, 124
                                           digi.xbee.util,457
digi.xbee.comm_interface, 464
                                           digi.xbee.util.utils, 457
digi.xbee.devices, 467
                                           digi.xbee.util.xmodem, 461
digi.xbee.exception, 934
                                           digi.xbee.xsocket, 1010
digi.xbee.filesystem, 937
digi.xbee.firmware, 954
digi.xbee.io,957
digi.xbee.models, 124
digi.xbee.models.accesspoint, 124
digi.xbee.models.address, 195
digi.xbee.models.atcomm, 126
digi.xbee.models.filesystem, 131
digi.xbee.models.hw, 187
digi.xbee.models.info, 190
digi.xbee.models.message, 199
digi.xbee.models.mode, 192
digi.xbee.models.options, 203
digi.xbee.models.protocol, 211
digi.xbee.models.status, 213
digi.xbee.models.zdo, 227
digi.xbee.packets, 236
digi.xbee.packets.aft, 236
digi.xbee.packets.base, 238
digi.xbee.packets.cellular, 247
digi.xbee.packets.common, 253
digi.xbee.packets.devicecloud, 299
digi.xbee.packets.digimesh, 318
digi.xbee.packets.factory, 455
digi.xbee.packets.filesystem, 323
digi.xbee.packets.network, 336
digi.xbee.packets.raw, 343
digi.xbee.packets.relay, 367
digi.xbee.packets.socket, 374
digi.xbee.packets.wifi,425
digi.xbee.packets.zigbee, 437
digi.xbee.profile, 963
digi.xbee.reader, 971
```

1020 Python Module Index

A	add_data_received_callback()
AbstractXBeeDevice (class in digi.xbee.devices), 467	(digi.xbee.devices.DigiMeshDevice method), 547
accept() (digi.xbee.xsocket.socket method), 1011 AccessPoint (class in digi.xbee.models.accesspoint), 124	<pre>add_data_received_callback()      (digi.xbee.devices.DigiPointDevice method),</pre>
<pre>ack_timeout_count      (digi.xbee.packets.digimesh.RouteInformationPac      attribute), 320</pre>	add_data_received_callback()
actual_offset (digi.xbee.models.filesystem.WriteFileCattribute), 151	097
ADC (digi.xbee.io.IOMode attribute), 962 add_bluetooth_data_received_callback() (digi.xbee.devices.CellularDevice method), 673	add_data_received_callback() (digi.xbee.devices.NBIoTDevice method), 723
<pre>add_bluetooth_data_received_callback()       (digi.xbee.devices.DigiMeshDevice method), 547</pre>	<pre>add_data_received_callback()      (digi.xbee.devices.Raw802Device method), 515</pre>
add_bluetooth_data_received_callback()	<pre>add_data_received_callback()           (digi.xbee.devices.WiFiDevice method), 755 add_data_received_callback()</pre>
add_bluetooth_data_received_callback() (digi.xbee.devices.IPDevice method), 647	(digi.xbee.devices.XBeeDevice method), 491
add_bluetooth_data_received_callback()	<pre>add_data_received_callback()       (digi.xbee.devices.ZigBeeDevice method), 616</pre>
(digi.xbee.devices.NBIoTDevice method), 722 add_bluetooth_data_received_callback() (digi.xbee.devices.Raw802Device method), 515	<pre>add_data_received_callback()      (digi.xbee.reader.PacketListener method),      994</pre>
add_bluetooth_data_received_callback() (digi.xbee.devices.WiFiDevice method), 755	<pre>add_device_discovered_callback()           (digi.xbee.devices.DigiMeshNetwork method),</pre>
<pre>add_bluetooth_data_received_callback()      (digi.xbee.devices.XBeeDevice method), 491</pre>	908 add_device_discovered_callback()
add_bluetooth_data_received_callback() (digi.xbee.devices.ZigBeeDevice method), 616	(digi.xbee.devices.DigiPointNetwork method), 919
<pre>add_bluetooth_data_received_callback()           (digi.xbee.reader.PacketListener method), 995 add_data_received_callback()</pre>	<pre>add_device_discovered_callback()      (digi.xbee.devices.Raw802Network method),      896</pre>
(digi.xbee.devices.CellularDevice method), 673	<pre>add_device_discovered_callback()       (digi.xbee.devices.XBeeNetwork method),       877</pre>

```
add device discovered callback()
                                                add_expl_data_received_callback()
        (digi.xbee.devices.ZigBeeNetwork
                                       method),
                                                         (digi.xbee.devices.XBeeDevice
                                                                                        method),
add_discovery_process_finished_callback()add_expl_data_received_callback()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                         (digi.xbee.devices.ZigBeeDevice
                                                                                        method),
add_discovery_process_finished_callback()add_explicit_data_received_callback()
        (digi.xbee.devices.DigiPointNetwork method),
                                                         (digi.xbee.reader.PacketListener method), 994
        919
                                                 add fs frame received callback()
                                                         (digi.xbee.devices.CellularDevice
add_discovery_process_finished_callback()
                                                                                        method),
        (digi.xbee.devices.Raw802Network
                                       method),
                                                add_fs_frame_received_callback()
                                                         (digi.xbee.devices.DigiMeshDevice
add_discovery_process_finished_callback()
                                                                                        method),
        (digi.xbee.devices.XBeeNetwork method), 878
                                                         547
add_discovery_process_finished_callback()add_fs_frame_received_callback()
        (digi.xbee.devices.ZigBeeNetwork
                                       method),
                                                         (digi.xbee.devices.DigiPointDevice
                                                                                        method),
        885
                                                         579
add_end_discovery_scan_callback()
                                                add fs frame received callback()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                         (digi.xbee.devices.IPDevice method), 647
                                                add fs frame received callback()
add_end_discovery_scan_callback()
                                                         (digi.xbee.devices.LPWANDevice
                                                                                        method),
        (digi.xbee.devices.DigiPointNetwork method),
        920
                                                add_fs_frame_received_callback()
add_end_discovery_scan_callback()
                                                         (digi.xbee.devices.NBIoTDevice
                                                                                        method).
                                                         723
        (digi.xbee.devices.Raw802Network
                                       method),
                                                add_fs_frame_received_callback()
add_end_discovery_scan_callback()
                                                         (digi.xbee.devices.Raw802Device
                                                                                        method),
        (digi.xbee.devices.XBeeNetwork
                                       method),
                                                add_fs_frame_received_callback()
add_end_discovery_scan_callback()
                                                         (digi.xbee.devices.WiFiDevice method), 755
        (digi.xbee.devices.ZigBeeNetwork
                                       method),
                                                add_fs_frame_received_callback()
        886
                                                         (digi.xbee.devices.XBeeDevice
                                                                                        method),
                                                         492
add_expl_data_received_callback()
        (digi.xbee.devices.CellularDevice
                                       method),
                                                add_fs_frame_received_callback()
                                                         (digi.xbee.devices.ZigBeeDevice
                                                                                        method),
add_expl_data_received_callback()
        (digi.xbee.devices.DigiMeshDevice
                                       method),
                                                add fs frame received callback()
                                                         (digi.xbee.reader.PacketListener
                                                                                        method),
                                                         996
add_expl_data_received_callback()
        (digi.xbee.devices.DigiPointDevice
                                       method),
                                                add_if_not_exist()
                                                         (digi.xbee.devices.DigiMeshNetwork method),
add_expl_data_received_callback()
                                                         909
        (digi.xbee.devices.IPDevice method), 647
                                                add if not exist()
                                                         (digi.xbee.devices.DigiPointNetwork method),
add_expl_data_received_callback()
        (digi.xbee.devices.LPWANDevice
                                       method),
        697
                                                add_if_not_exist()
add_expl_data_received_callback()
                                                         (digi.xbee.devices.Raw802Network
                                                                                        method),
        (digi.xbee.devices.NBIoTDevice
                                                         897
                                       method),
                                                add_if_not_exist()
add_expl_data_received_callback()
                                                         (digi.xbee.devices.XBeeNetwork
                                                                                        method),
                                       method),
                                                         883
        (digi.xbee.devices.Raw802Device
                                                add if not exist()
add_expl_data_received_callback()
                                                         (digi.xbee.devices.ZigBeeNetwork
                                                                                        method),
        (digi.xbee.devices.WiFiDevice method), 755
                                                         886
```

add\_init\_discovery\_scan\_callback() (digi.xbee.devices.NBIoTDevice method), (digi.xbee.devices.DigiMeshNetwork method), 723 add\_ip\_data\_received\_callback() (digi.xbee.devices.WiFiDevice method), 755 add\_init\_discovery\_scan\_callback() (digi.xbee.devices.DigiPointNetwork method), add\_ip\_data\_received\_callback() (digi.xbee.reader.PacketListener method), 995 add\_init\_discovery\_scan\_callback() (digi.xbee.devices.Raw802Network method), add\_micropython\_data\_received\_callback() (digi.xbee.devices.CellularDevice method), 673 898 add\_init\_discovery\_scan\_callback() add\_micropython\_data\_received\_callback() (digi.xbee.devices.XBeeNetwork method), 877 (digi.xbee.devices.DigiMeshDevice method), add\_init\_discovery\_scan\_callback() add\_micropython\_data\_received\_callback() (digi.xbee.devices.ZigBeeNetwork method), 886 (digi.xbee.devices.DigiPointDevice add\_io\_sample\_received\_callback() (digi.xbee.devices.CellularDevice method), add\_micropython\_data\_received\_callback() (digi.xbee.devices.IPDevice method), 648 add\_io\_sample\_received\_callback() add\_micropython\_data\_received\_callback() method), (digi.xbee.devices.DigiMeshDevice (digi.xbee.devices.LPWANDevice method), 698 add\_micropython\_data\_received\_callback() add\_io\_sample\_received\_callback() (digi.xbee.devices.NBIoTDevice method), 723 (digi.xbee.devices.DigiPointDevice method), add\_micropython\_data\_received\_callback() 579 (digi.xbee.devices.Raw802Device method), 516 add io sample received callback() add micropython data received callback() (digi.xbee.devices.IPDevice method), 648 (digi.xbee.devices.WiFiDevice method), 755 add\_io\_sample\_received\_callback() add\_micropython\_data\_received\_callback() (digi.xbee.devices.LPWANDevice method), (digi.xbee.devices.XBeeDevice method), 491 add\_micropython\_data\_received\_callback() (digi.xbee.devices.ZigBeeDevice method), 617 add\_io\_sample\_received\_callback() (digi.xbee.devices.NBIoTDevice method), add\_micropython\_data\_received\_callback() (digi.xbee.reader.PacketListener method), 995 add\_io\_sample\_received\_callback() add\_modem\_status\_received\_callback() (digi.xbee.devices.Raw802Device method), (digi.xbee.devices.CellularDevice method), 674 add\_modem\_status\_received\_callback() 515 add\_io\_sample\_received\_callback() (digi.xbee.devices.DigiMeshDevice (digi.xbee.devices.WiFiDevice method), 755 add\_io\_sample\_received\_callback() add modem status received callback() (digi.xbee.devices.XBeeDevice method), (digi.xbee.devices.DigiPointDevice method), 491 579 add\_io\_sample\_received\_callback() add\_modem\_status\_received\_callback() (digi.xbee.devices.ZigBeeDevice method), (digi.xbee.devices.IPDevice method), 648 add\_modem\_status\_received\_callback() (digi.xbee.devices.LPWANDevice method), 698 add\_io\_sample\_received\_callback() (digi.xbee.reader.PacketListener method), add\_modem\_status\_received\_callback() (digi.xbee.devices.NBIoTDevice method), 723 add\_ip\_data\_received\_callback() add\_modem\_status\_received\_callback() (digi.xbee.devices.CellularDevice method), (digi.xbee.devices.Raw802Device method), 516 673 add\_modem\_status\_received\_callback() add\_ip\_data\_received\_callback() (digi.xbee.devices.WiFiDevice method), 756 (digi.xbee.devices.IPDevice method), 644 add\_modem\_status\_received\_callback() add\_ip\_data\_received\_callback() (digi.xbee.devices.XBeeDevice method), 491 (digi.xbee.devices.LPWANDevice *method*), add modem status received callback() 698 (digi.xbee.devices.ZigBeeDevice method), 617 add\_ip\_data\_received\_callback() add\_modem\_status\_received\_callback()

(digi.xbee.reader.PacketListener method), 994	(digi.xbee.devices.Raw802Network method),
$\begin{tabular}{ll} add\_network\_modified\_callback()\\ ({\it digi.xbee.devices.DigiMeshNetwork}\ {\it method}), \end{tabular}$	898 add_packet_received_from_callback()
909	(digi.xbee.devices.XBeeNetwork method), 878
<pre>add_network_modified_callback()</pre>	<pre>add_packet_received_from_callback()</pre>
(digi.xbee.devices.DigiPointNetwork method), 921	(digi.xbee.devices.ZigBeeNetwork method), 887
<pre>add_network_modified_callback()</pre>	add_packet_received_from_callback()
(digi.xbee.devices.Raw802Network method), 898	(digi.xbee.reader.PacketListener method), 994 add_remote() (digi.xbee.devices.DigiMeshNetwork
<pre>add_network_modified_callback()</pre>	method), 910
(digi.xbee.devices.XBeeNetwork method), 876	add_remote() (digi.xbee.devices.DigiPointNetwork method), 921
add_network_modified_callback()	add_remote() (digi.xbee.devices.Raw802Network
(digi.xbee.devices.ZigBeeNetwork method),	method), 898
886	add_remote() (digi.xbee.devices.XBeeNetwork
<pre>add_packet_received_callback()</pre>	method), 884
(digi.xbee.devices.CellularDevice method),	add_remote() (digi.xbee.devices.ZigBeeNetwork
674	method), 887
<pre>add_packet_received_callback()</pre>	add_remotes() (digi.xbee.devices.DigiMeshNetwork
(digi.xbee.devices.DigiMeshDevice method),	method), 910
548	add_remotes() (digi.xbee.devices.DigiPointNetwork
<pre>add_packet_received_callback()</pre>	method), 921
(digi.xbee.devices.DigiPointDevice method), 580	add_remotes() (digi.xbee.devices.Raw802Network method), 899
<pre>add_packet_received_callback()      (digi.xbee.devices.IPDevice method), 648</pre>	add_remotes() (digi.xbee.devices.XBeeNetwork method), 884
add_packet_received_callback()	add_remotes() (digi.xbee.devices.ZigBeeNetwork
(digi.xbee.devices.LPWANDevice method),	method), 887
698	add_route_info_received_callback()
<pre>add_packet_received_callback()</pre>	(digi.xbee.reader.PacketListener method), 996
(digi.xbee.devices.NBIoTDevice method),	add_route_received_callback()
723	(digi.xbee.devices.CellularDevice method),
<pre>add_packet_received_callback()</pre>	674
(digi.xbee.devices.Raw802Device method),	add_route_received_callback()
516	(digi.xbee.devices.DigiMeshDevice method),
<pre>add_packet_received_callback()      (digi.xbee.devices.WiFiDevice method), 756</pre>	548
add_packet_received_callback()	<pre>add_route_received_callback()       (digi.xbee.devices.DigiPointDevice method),</pre>
(digi.xbee.devices.XBeeDevice method),	580
491	add_route_received_callback()
add_packet_received_callback()	(digi.xbee.devices.IPDevice method), 648
(digi.xbee.devices.ZigBeeDevice method),	add_route_received_callback()
617	(digi.xbee.devices.LPWANDevice method),
add_packet_received_callback()	698
(digi.xbee.reader.PacketListener method),	add_route_received_callback()
994	(digi.xbee.devices.NBIoTDevice method),
	(digi.xbec.devices.ivBio1Device memoa),
add_packet_received_from_callback()	724
<pre>add_packet_received_from_callback()      (digi.xbee.devices.DigiMeshNetwork method),      909</pre>	724
<pre>add_packet_received_from_callback()           (digi.xbee.devices.DigiMeshNetwork method),           909 add_packet_received_from_callback()</pre>	724 add_route_received_callback() (digi.xbee.devices.Raw802Device method), 516
<pre>add_packet_received_from_callback()      (digi.xbee.devices.DigiMeshNetwork method),      909</pre>	724 add_route_received_callback() (digi.xbee.devices.Raw802Device method),
<pre>add_packet_received_from_callback()           (digi.xbee.devices.DigiMeshNetwork method),           909 add_packet_received_from_callback()           (digi.xbee.devices.DigiPointNetwork method),</pre>	724 add_route_received_callback()

```
(digi.xbee.devices.XBeeDevice
                                       method),
                                                        (digi.xbee.devices.LPWANDevice method), 699
       495
                                                add_socket_data_received_from_callback()
                                                        (digi.xbee.devices.NBIoTDevice method), 724
add route received callback()
        (digi.xbee.devices.ZigBeeDevice
                                                add_socket_data_received_from_callback()
                                       method),
                                                        (digi.xbee.devices.Raw802Device method), 516
                                                add socket data received from callback()
add route record received callback()
        (digi.xbee.reader.PacketListener method), 996
                                                        (digi.xbee.devices.WiFiDevice method), 756
                                                add_socket_data_received_from_callback()
add sms callback()
        (digi.xbee.devices.CellularDevice
                                       method),
                                                        (digi.xbee.devices.XBeeDevice method), 492
        670
                                                add_socket_data_received_from_callback()
add_sms_callback()
                                                        (digi.xbee.devices.ZigBeeDevice method), 618
        (digi.xbee.devices.LPWANDevice
                                                add_socket_data_received_from_callback()
                                       method),
                                                        (digi.xbee.reader.PacketListener method), 996
add_sms_callback()
                                                add_socket_state_callback()
        (digi.xbee.devices.NBIoTDevice
                                       method),
                                                        (digi.xbee.xsocket.socket method), 1014
        724
                                                add_socket_state_received_callback()
                                                        (digi.xbee.devices.CellularDevice method), 674
add_sms_received_callback()
        (digi.xbee.reader.PacketListener
                                                add socket state received callback()
                                       method),
                                                        (digi.xbee.devices.DigiMeshDevice
                                                                                       method),
add_socket_data_received_callback()
        (digi.xbee.devices.CellularDevice method), 674
                                                add_socket_state_received_callback()
add_socket_data_received_callback()
                                                        (digi.xbee.devices.DigiPointDevice
        (digi.xbee.devices.DigiMeshDevice
                                       method),
                                                        580
                                                add_socket_state_received_callback()
add_socket_data_received_callback()
                                                        (digi.xbee.devices.IPDevice method), 649
        (digi.xbee.devices.DigiPointDevice
                                       method),
                                                add_socket_state_received_callback()
                                                        (digi.xbee.devices.LPWANDevice method), 699
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.IPDevice method), 649
                                                        (digi.xbee.devices.NBIoTDevice method), 724
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.LPWANDevice method), 699
                                                        (digi.xbee.devices.Raw802Device method), 517
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.NBIoTDevice method), 724
                                                        (digi.xbee.devices.WiFiDevice method), 756
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.Raw802Device method), 516
                                                        (digi.xbee.devices.XBeeDevice method), 492
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.WiFiDevice method), 756
                                                        (digi.xbee.devices.ZigBeeDevice method), 618
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.XBeeDevice method), 492
                                                        (digi.xbee.reader.PacketListener method), 995
add_socket_data_received_callback()
                                                add_user_data_relay_received_callback()
        (digi.xbee.devices.ZigBeeDevice method), 617
                                                        (digi.xbee.devices.CellularDevice method), 675
add_socket_data_received_callback()
                                                add_user_data_relay_received_callback()
        (digi.xbee.reader.PacketListener method), 995
                                                        (digi.xbee.devices.DigiMeshDevice
add_socket_data_received_from_callback()
        (digi.xbee.devices.CellularDevice method), 674
                                                add_user_data_relay_received_callback()
add_socket_data_received_from_callback()
                                                        (digi.xbee.devices.DigiPointDevice
                                                                                       method),
        (digi.xbee.devices.DigiMeshDevice
                                       method),
                                                        580
        549
                                                add_user_data_relay_received_callback()
add_socket_data_received_from_callback()
                                                        (digi.xbee.devices.IPDevice method), 649
        (digi.xbee.devices.DigiPointDevice
                                       method), add_user_data_relay_received_callback()
                                                        (digi.xbee.devices.LPWANDevice method), 699
add_socket_data_received_from_callback()add_user_data_relay_received_callback()
        (digi.xbee.devices.IPDevice method), 649
                                                        (digi.xbee.devices.NBIoTDevice method), 724
add_socket_data_received_from_callback()add_user_data_relay_received_callback()
```

(digi.xbee.devices.Raw802Device method), 517 (digi.xbee.reader.RelayDataReceived append() add\_user\_data\_relay\_received\_callback() method), 982 (digi.xbee.devices.WiFiDevice method), 756 append() (digi.xbee.reader.RouteInformationReceived add\_user\_data\_relay\_received\_callback() method), 988 (digi.xbee.devices.XBeeDevice method), 491 append() (digi.xbee.reader.RouteReceived method), add user data relay received callback() 989 append() (digi.xbee.reader.RouteRecordIndicatorReceived (digi.xbee.devices.ZigBeeDevice method), 618 add\_user\_data\_relay\_received\_callback() method), 987 (digi.xbee.reader.PacketListener method), 995 append() (digi.xbee.reader.SMSReceived method), 981 (digi.xbee.reader.SocketDataReceived address (digi.xbee.models.address.XBee16BitAddress append() attribute), 197 method), 985 address (digi.xbee.models.address.XBee64BitAddress (digi.xbee.reader.SocketDataReceivedFrom append() attribute), 198 method), 986 append() address (digi.xbee.models.address.XBeeIMEIAddress (digi.xbee.reader.SocketStateReceived attribute), 198 method), 985 analog\_mask (digi.xbee.io.IOSample attribute), 960 append () (digi.xbee.reader.XBeeEvent method), 972 analog\_values (digi.xbee.io.IOSample attribute), APPEND\_DD (digi.xbee.models.options.DiscoveryOptions 960 attribute), 208 ApiFrameType (class in digi.xbee.packets.aft), 236 APPEND\_RSSI (digi.xbee.models.options.DiscoveryOptions APIOutputMode (class in digi.xbee.models.mode), 192 attribute), 208 APIOutputModeBit (class in digi.xbee.models.mode), APPLY\_CHANGES (digi.xbee.models.options.RemoteATCmdOptions attribute), 207  $\verb|APPEND| (digi.xbee.models.options.FileOpenRequestOptionapply\_changes()| (digi.xbee.devices.AbstractXBeeDevices)| (digi.xbee.devices)| (digi.xbee.devices)$ method), 469 attribute), 211 append() (digi.xbee.reader.BluetoothDataReceived apply\_changes() (digi.xbee.devices.CellularDevice *method*), 983 method), 675 (digi.xbee.reader.DataReceived method), apply\_changes()(digi.xbee.devices.DigiMeshDevice append() method), 549 append() (digi.xbee.reader.DeviceDiscovered method), apply\_changes() (digi.xbee.devices.DigiPointDevice 978 method), 580 append() (digi.xbee.reader.DiscoveryProcessFinished apply\_changes() (digi.xbee.devices.IPDevice method), 979 method), 649 (digi.xbee.reader.EndDiscoveryScan apply\_changes() (digi.xbee.devices.LPWANDevice append() method), 991 method), 699 apply\_changes() (digi.xbee.reader.ExplicitDataReceived (digi.xbee.devices.NBIoTDevice append() method), 724 *method*), 980 append() (digi.xbee.reader.FileSystemFrameReceived apply\_changes() (digi.xbee.devices.Raw802Device method), 992 method), 517 apply\_changes() (digi.xbee.devices.RemoteDigiMeshDevice (digi.xbee.reader.InitDiscoveryScan append() method), 990 *method*), 818 (digi.xbee.reader.IOSampleReceived  $apply\_changes()$  (digi.xbee.devices.RemoteDigiPointDevice append() *method*), 976 method), 836 append() (digi.xbee.reader.IPDataReceived method), apply\_changes()(digi.xbee.devices.RemoteRaw802Device 980 method), 799 append() (digi.xbee.reader.MicroPythonDataReceived apply\_changes() (digi.xbee.devices.RemoteXBeeDevice method), 984 method), 783  $\verb"apply_changes" () \textit{ (digi.xbee.devices.RemoteZigBeeDevice}$ append() (digi.xbee.reader.ModemStatusReceived *method*), 975 *method*), 856 append() (digi.xbee.reader.NetworkModified method), apply\_changes() (digi.xbee.devices.WiFiDevice method), 757 append() (digi.xbee.reader.PacketReceived method), apply\_changes() (digi.xbee.devices.XBeeDevice 972 method), 496 (digi.xbee.reader.PacketReceivedFrom apply\_changes() (digi.xbee.devices.ZigBeeDevice append() *method*), 973 method), 618

<pre>apply_profile() (digi.xbee.comm_interface.XBeeCom</pre>	nmunicatio <b>ù 5</b> 8terface
method), 466	ATCommQueuePacket (class in
<pre>apply_profile() (digi.xbee.devices.AbstractXBeeDev</pre>	
method), 483	ATCommResponsePacket (class in
apply_profile() (digi.xbee.devices.CellularDevice	digi.xbee.packets.common), 259
method), 675	ATStringCommand (class in
apply_profile()(digi.xbee.devices.DigiMeshDevice	digi.xbee.models.atcomm), 126
method), 549	В
<pre>apply_profile() (digi.xbee.devices.DigiPointDevice</pre>	
apply_profile() (digi.xbee.devices.IPDevice	baudrate (digi.xbee.profile.FirmwareBaudrate at-
method), 649	tribute), 963
apply_profile() (digi.xbee.devices.LPWANDevice	bind() (digi.xbee.xsocket.socket method), 1010
method), 699	block_number (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacattribute), 455
apply_profile() (digi.xbee.devices.NBIoTDevice	block_size (digi.xbee.filesystem.FileProcess at-
method), 725	tribute), 938
apply_profile() (digi.xbee.devices.Raw802Device	BluetoothDataReceived (class in
method), 517	digi.xbee.reader), 983
$\verb"apply_profile" () \textit{ (digi.xbee.devices.RemoteDigiMesh.} \\$	Periseloader file (digi.xbee.profile.XBeeProfile at-
method), 818	tribute), 971
$\verb"apply_profile" () \textit{ (digi.xbee.devices.RemoteDigiPoint} \\$	Dgyi6€loader_msg_type
method), 836	(digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
apply_profile() (digi.xbee.devices.RemoteRaw802D	evice attribute), 455
method), 799	BROADCAST_ADDRESS
<pre>apply_profile() (digi.xbee.devices.RemoteXBeeDevi</pre>	ce (digi.xbee.models.address.XBee16BitAddress
method), 783	attribute), 195
apply_profile()(digi.xbee.devices.RemoteZigBeeDe	
method), 856	(digi.x bee.models. address. XBee 64Bit Address
apply_profile() (digi.xbee.devices.WiFiDevice	attribute), 197
method), 757	BROADCAST_PACKET (digi.xbee.models.options.ReceiveOptions
apply_profile() (digi.xbee.devices.XBeeDevice	attribute), 204
<pre>method), 496 apply_profile() (digi.xbee.devices.ZigBeeDevice</pre>	BROADCAST_PAN (digi.xbee.models.options.TransmitOptions
method), 618	attribute), 205
apply_profile() (digi.xbee.serial.XBeeSerialPort	BROADCAST_PANS_PACKET
method), 1007	(digi.xbee.models.options.ReceiveOptions attribute), 204
apply_xbee_profile() (in module	broadcast_radius(digi.xbee.packets.common.ExplicitAddressingPack
digi.xbee.profile), 971	attribute), 294
	Moroadcast_radius(digi.xbee.packets.common.TransmitPacket
attribute), 204	attribute), 277
<pre>ascii_to_int() (in module digi.xbee.util.utils), 459</pre>	build_aggregate_routes()
AssociationIndicationStatus (class in	(digi.xbee.devices.DigiMeshDevice method),
digi.xbee.models.status), 218	542
at_command (digi.xbee.io.IOLine attribute), 958	<pre>build_frame() (in module digi.xbee.packets.factory),</pre>
at_response_received_cb()	457
(digi.xbee.sender.PacketSender method),	<pre>build_fs_command()</pre>
1005	digi.xbee.packets.filesystem), 336
ATCommand (class in digi.xbee.models.atcomm), 129	bytearray_value(digi.xbee.profile.XBeeProfileSetting
ATCommandException, 934	attribute), 967
ATCommandResponse (class in	<pre>bytes_bad(digi.xbee.models.filesystem.VolFormatCmdResponse</pre>
digi.xbee.models.atcomm), 130	attribute), 186
ATCommandStatus (class in digi.xbee.models.status),	$\verb bytes_bad  (\textit{digi.xbee.models.filesystem.VolStatCmdResponse} $
213	attribute), 182
ATCommPacket (class in digi.xbee.packets.common),	

bytes_free(digi.xbee.models.filesystem.VolFormatCmo attribute), 186	dRexpanse() (digi.xbee.reader.ModemStatusReceived method), 975
bytes_free(digi.xbee.models.filesystem.VolStatCmdRe attribute), 182	
bytes_to_int() (in module digi.xbee.util.utils), 459	
bytes_used(digi.xbee.models.filesystem.VolFormatCmatCmatCmatCmatCmatCmatCmatCmatCmatC	
attribute), 186	clear() (digi.xbee.reader.PacketReceivedFrom
bytes_used(digi.xbee.models.filesystem.VolStatCmdRe	
attribute), 182	clear() (digi.xbee.reader.RelayDataReceived
	method), 982
C	clear() (digi.xbee.reader.RouteInformationReceived
CASCADE (digi.xbee.models.mode.NeighborDiscoveryMod	J. D. 000
attribute), 194	clear() (digi.xbee.reader.RouteReceived method), 989
CellularAssociationIndicationStatus	clear() (digi.xbee.reader.RouteRecordIndicatorReceived
(class in digi.xbee.models.status), 220	method), 987
CellularDevice (class in digi.xbee.devices), 669	clear() (digi.xbee.reader.SMSReceived method), 981
change_directory()	clear() (digi.xbee.reader.SocketDataReceived
(digi.xbee.filesystem.LocalXBeeFileSystemMana	ger method), 985
method), 951	clear() (digi.xbee.reader.SocketDataReceivedFrom
channel (digi.xbee.models.accesspoint.AccessPoint at-	method), 986
tribute), 125	clear() (digi.xbee.reader.SocketStateReceived
check_fs_support() (in module	method), 985
digi.xbee.filesystem), 954	clear() (digi.xbee.reader.XBeeEvent method), 972
<pre>clear() (digi.xbee.devices.DigiMeshNetwork method),</pre>	client_socket_id(digi.xbee.packets.socket.SocketNewIPv4ClientPac
910	attribute), 414
clear() (digi.xbee.devices.DigiPointNetwork method), 921	close() (digi.xbee.comm_interface.XBeeCommunicationInterface method), 464
clear() (digi.xbee.devices.Raw802Network method), 899	close() (digi.xbee.devices.CellularDevice method), 675
clear() (digi.xbee.devices.XBeeNetwork method), 879	close() (digi.xbee.devices.DigiMeshDevice method),
<pre>clear() (digi.xbee.devices.ZigBeeNetwork method),</pre>	550
887	close() (digi.xbee.devices.DigiPointDevice method),
clear() (digi.xbee.reader.BluetoothDataReceived	581
method), 983	close() (digi.xbee.devices.IPDevice method), 650
clear() (digi.xbee.reader.DataReceived method), 974	close() (digi.xbee.devices.LPWANDevice method),
clear() (digi.xbee.reader.DeviceDiscovered method),	700
978	close() (digi.xbee.devices.NBIoTDevice method), 725 close() (digi.xbee.devices.Raw802Device method),
clear() (digi.xbee.reader.DiscoveryProcessFinished method), 979	517
clear() (digi.xbee.reader.EndDiscoveryScan method),	close() (digi.xbee.devices.WiFiDevice method), 757
991	close() (digi.xbee.devices.XBeeDevice method), 486
clear() (digi.xbee.reader.ExplicitDataReceived	close() (digi.xbee.devices.ZigBeeDevice method), 619 close() (digi.xbee.profile.XBeeProfile method), 968
method), 980	close() (digi.xbee.serial.XBeeSerialPort method),
clear() (digi.xbee.reader.FileSystemFrameReceived method), 992	1008
clear() (digi.xbee.reader.InitDiscoveryScan method),	close() (digi.xbee.xsocket.socket method), 1013
990	CloseDirCmdRequest (class in
clear() (digi.xbee.reader.IOSampleReceived method),	digi.xbee.models.filesystem), 163
976	CloseDirCmdResponse (class in digi.xbee.models.filesystem), 164
clear() (digi.xbee.reader.IPDataReceived method),	closeFileCmdRequest (class in
980	digi.xbee.models.filesystem), 142
clear() (digi.xbee.reader.MicroPythonDataReceived	CloseFileCmdResponse (class in
method), 984	digi.xbee.models.filesystem), 143
	arguino como word juco jorem), 115

cluster\_id(digi.xbee.models.message.ExplicitXBeeMessage tribute), 214 code (digi.xbee.models.status.EmberBootloaderMessageType attribute), 200 cluster id (digi.xbee.packets.common.ExplicitAddressingPacket attribute), 224 attribute), 294 code (digi.xbee.models.status.FrameError attribute), cluster\_id(digi.xbee.packets.common.ExplicitRXIndicatorPacket222 attribute), 299 (digi.xbee.models.status.FSCommandStatus code code (digi.xbee.devices.NetworkEventReason attribute), attribute), 226 931 code (digi.xbee.models.status.ModemStatus attribute), code (digi.xbee.devices.NetworkEventType attribute), 217 931 code (digi.xbee.models.status.NetworkDiscoveryStatus code (digi.xbee.io.IOValue attribute), 958 attribute), 223 code (digi.xbee.models.accesspoint.WiFiEncryptionType (digi.xbee.models.status.PowerLevel attribute), code attribute), 126 code (digi.xbee.models.atcomm.SpecialByte attribute), code (digi.xbee.models.status.SocketInfoState attribute), 226 code (digi.xbee.models.filesystem.FSCmdType code (digi.xbee.models.status.SocketState attribute), 225 tribute), 131 code (digi.xbee.models.status.SocketStatus attribute), code (digi.xbee.models.hw.HardwareVersion attribute), 224 code (digi.xbee.models.status.TransmitStatus attribute), code (digi.xbee.models.hw.LegacyHardwareVersion attribute), 190  $\verb"code" (digi.xbee.models.status.WiFiAssociationIndicationStatus$ (digi.xbee.models.mode.APIOutputMode attribute), 222 code atcode (digi.xbee.models.status.ZigbeeRegisterStatus attribute), 193 (digi.xbee.models.mode.APIOutputModeBit code tribute), 223 attribute), 193 code (digi.xbee.packets.aft.ApiFrameType attribute), code (digi.xbee.models.mode.IPAddressingMode tribute), 194 code (digi.xbee.profile.FlashFirmwareOption attribute), (digi.xbee.models.mode.Neighbor Discovery Mode965 code attribute), 194 comm\_iface (digi.xbee.devices.CellularDevice code (digi.xbee.models.mode.OperatingMode attribute), tribute), 675 comm\_iface (digi.xbee.devices.DigiMeshDevice atcode (digi.xbee.models.options.DiscoveryOptions attribute), 550 tribute), 208 comm\_iface (digi.xbee.devices.DigiPointDevice code (digi.xbee.models.options.RegisterKeyOptions atattribute), 581 tribute), 209 (digi.xbee.devices.IPDevice attribute), comm iface  $\verb"code" (digi.x bee.models.options. Send Data Request Options$ 650 attribute), 207 comm iface (digi.xbee.devices.LPWANDevice code (digi.xbee.models.options.SocketOption attribute), tribute), 700 210 comm iface (digi.xbee.devices.NBIoTDevice atcode (digi.xbee.models.options.XBeeLocalInterface at*tribute*), 725 (digi.xbee.devices.Raw802Device tribute), 209 comm iface code (digi.xbee.models.protocol.IPProtocol attribute), tribute), 517 comm\_iface (digi.xbee.devices.WiFiDevice attribute), (digi.xbee.models.protocol.XBeeProtocolcode at-757 tribute), 212 comm\_iface (digi.xbee.devices.XBeeDevice attribute), code (digi.xbee.models.status.AssociationIndicationStatus 486 attribute), 220 comm iface (digi.xbee.devices.ZigBeeDevice (digi.xbee.models.status.ATCommandStatus code tribute), 619 attribute), 214 command (digi.xbee.models.atcomm.ATCommand atcode (digi.xbee.models.status.CellularAssociationIndicationStatus tribute), 130 attribute), 220 command (digi.xbee.models.atcomm.ATCommandResponse code (digi.xbee.models.status.DeviceCloudStatus attribute), 130 attribute), 221 command (digi.xbee.models.atcomm.ATStringCommand (digi.xbee.models.status.DiscoveryStatus attribute), 129

Index 1029

at-

code

		(digi.xbee.reader.DataReceived method), 974
attribute), 255		(digi.xbee.reader. Device Discovered  method),
$\verb command  (digi.xbee.packets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.common.ATCommQueuePackets.com.com.com.com.com.com.com.com.com.com$		978
attribute), 258	copy()	(digi.xbee.reader.DiscoveryProcessFinished
$\verb command  (\textit{digi.xbee.packets.common.ATCommResponse})                                      $		method), 979
attribute), 261		(digi.xbee.reader.EndDiscoveryScan method),
command (digi.xbee.packets.common.RemoteATComma		991
attribute), 269	copy()	(digi.xbee.reader.ExplicitDataReceived
command (digi.xbee.packets.common.RemoteATComma	-	
attribute), 272	copy()	(digi.xbee.reader.FileSystemFrameReceived
command (digi.xbee.packets.filesystem.FSRequestPacke		method), 992
attribute), 324	copy()	
command (digi.xbee.packets.filesystem.FSResponsePack		990
attribute), 327	copy()	
command (digi.xbee.packets.filesystem.RemoteFSReque		976
attribute), 330		(digi.xbee.reader.IPDataReceived method), 980
command (digi.xbee.packets.filesystem.RemoteFSRespon	nsercacyper ()	(digi.xbee.reader.MicroPythonDataReceived
attribute), 335	an ama dili GV	method), 984
command (digi.xbee.packets.wifi.RemoteATCommandReattribute), 436	espousevyijiF	method), 975
command (digi.xbee.packets.wifi.RemoteATCommandW	ifi Parakast ; ()	
attribute), 431	ijii acaeiy ()	977
command_value (digi.xbee.packets.common.ATComm	Dagmon sa Da	
attribute), 261		(digi.xbee.reader.PacketReceivedFrom method),
command_value (digi.xbee.packets.common.RemoteA		
attribute), 273		(digi.xbee.reader.RelayDataReceived method),
command_value(digi.xbee.packets.wifi.RemoteATCon		
attribute), 436	<i>с</i> ору ()	(digi.xbee.reader.RouteInformationReceived
CommunicationException, 934	сору ()	method), 988
compatibility_number	copy()	(digi.xbee.reader.RouteReceived method), 989
(digi.xbee.profile.XBeeProfile attribute)		(digi.xbee.reader.RouteRecordIndicatorReceived
970	, 551 ()	method), 987
complex_desc_supported	() yang	(digi.xbee.reader.SMSReceived method), 981
(digi.xbee.models.zdo.NodeDescriptor at		(digi.xbee.reader.SocketDataReceived method),
tribute), 228	12 (/	986
connect () (digi.xbee.filesystem.LocalXBeeFileSystem	Managev ()	(digi.xbee.reader.SocketDataReceivedFrom
method), 951	<i>a 1</i> · · ·	method), 986
connect () (digi.xbee.xsocket.socket method), 1010	copy()	(digi.xbee.reader.SocketStateReceived method),
connect_by_ap() (digi.xbee.devices.WiFiDevice		985
method), 750		(digi.xbee.reader.XBeeEvent method), 972
connect_by_ssid() (digi.xbee.devices.WiFiDevice		
method), 750		method), 983
Connection (class in digi.xbee.devices), 932	count (	) (digi.xbee.reader.DataReceived method), 974
ConnectionException, 934		) (digi.xbee.reader.DeviceDiscovered method),
content_type (digi.xbee.packets.devicecloud.SendDe	ataRequestPo	$a\partial k e$
attribute), 313	count (	$) \hspace{0.5cm} (digi.xbee.reader. Discovery Process Finished \\$
COORDINATOR_ADDRESS		method), 979
(digi.x bee.models. address. XBee 16 Bit Address	count (	$) \ (digi.xbee.reader. End Discovery Scan\ method),$
attribute), 195		991
COORDINATOR_ADDRESS	count (	$) \\ (digi.xbee.reader. Explicit Data Received$
(digi.x bee.models. address. XBee 64Bit Address		method), 980
attribute), 197	count (	$) \qquad (digi.xbee.reader. File System Frame Received$
copy() (digi.xbee.reader.BluetoothDataReceived		method), 992
method), 983	count (	) (digi.xbee.reader.InitDiscoveryScan method),

- 990 create\_cmd() (digi.xbee.models.filesystem.GetPathIdCmdRequest count () (digi.xbee.reader.IOSampleReceived method), class method), 170 create cmd() (digi.xbee.models.filesystem.GetPathIdCmdResponse count () (digi.xbee.reader.IPDataReceived method), class method), 172 create\_cmd() (digi.xbee.models.filesystem.HashFileCmdRequest count () (digi.xbee.reader.MicroPythonDataReceived class method), 153 method), 984 create cmd() (digi.xbee.models.filesystem.HashFileCmdResponse count() (digi.xbee.reader.ModemStatusReceived class method), 155 method), 975create\_cmd() (digi.xbee.models.filesystem.OpenDirCmdRequest count() (digi.xbee.reader.NetworkModified method), class method), 159 977 create\_cmd() (digi.xbee.models.filesystem.OpenDirCmdResponse count () (digi.xbee.reader.PacketReceived method), class method), 162 create\_cmd() (digi.xbee.models.filesystem.OpenFileCmdRequest (digi.xbee.reader.PacketReceivedFrom class method), 139 count() create\_cmd() (digi.xbee.models.filesystem.OpenFileCmdResponse method), 973 (digi.xbee.reader.RelayDataReceived class method), 141 count() method), 982 create\_cmd() (digi.xbee.models.filesystem.ReadDirCmdRequest (digi.xbee.reader.RouteInformationReceived class method), 166 count() method), 988 create cmd() (digi.xbee.models.filesystem.ReadDirCmdResponse count () (digi.xbee.reader.RouteReceived method), 989 class method), 168 count() (digi.xbee.reader.RouteRecordIndicatorReceivedcreate\_cmd() (digi.xbee.models.filesystem.ReadFileCmdRequest method), 987 class method), 146 count () (digi.xbee.reader.SMSReceived method), 981 create\_cmd() (digi.xbee.models.filesystem.ReadFileCmdResponse (digi.xbee.reader.SocketDataReceived class method), 147 count() method), 986 create\_cmd() (digi.xbee.models.filesystem.RenameCmdRequest count() (digi.xbee.reader.SocketDataReceivedFrom class method), 174 method), 986 create\_cmd() (digi.xbee.models.filesystem.RenameCmdResponse (digi.xbee.reader.SocketStateReceived class method), 176 count() *method*), 985 create\_cmd() (digi.xbee.models.filesystem.UnknownFSCmd count () (digi.xbee.reader.XBeeEvent method), 972 class method), 134 CREATE (digi.xbee.models.options.FileOpenRequestOptioncreate\_cmd() (digi.xbee.models.filesystem.VolFormatCmdRequest attribute), 210 class method), 185
  - attribute), 210 class method), 185 create\_cmd() (digi.xbee.models.filesystem.CloseDirCmdReequest\_cmd() (digi.xbee.models.filesystem.VolFormatCmdResponse
  - class method), 163 class method), 187

    create\_cmd() (digi.xbee.models.filesystem.CloseDirCmdResponse\_cmd() (digi.xbee.models.filesystem.VolStatCmdRequest
  - $class\ method),\ 180$   $\verb|create_cmd()|\ (digi.xbee.models.filesystem.CloseFileCmdRecatest_cmd()|\ (digi.xbee.models.filesystem.VolStatCmdResponse)|$
  - class method), 142 class method), 182
    create\_cmd() (digi.xbee.models.filesystem.CloseFileCmdResponsecmd() (digi.xbee.models.filesystem.WriteFileCmdRequest class method), 144 class method), 149
  - create\_cmd() (digi.xbee.models.filesystem.CreateDirCmdRequest\_cmd() (digi.xbee.models.filesystem.WriteFileCmdResponse class method), 156 class method), 151
  - create\_cmd() (digi.xbee.models.filesystem.CreateDirCndResponsepacket() (digi.xbee.packets.base.GenericXBeePacket class method), 158 static method), 242
  - create\_cmd() (digi.xbee.models.filesystem.DeleteCmdRequestte\_packet() (digi.xbee.packets.base.UnknownXBeePacket class method), 177 static method), 245
  - create\_cmd() (digi.xbee.models.filesystem.DeleteCmdResponsee\_packet() (digi.xbee.packets.base.XBeeAPIPacket class method), 179 static method), 241
  - create\_cmd() (digi.xbee.models.filesystem.FileIdCmd create\_packet() (digi.xbee.packets.base.XBeePacket class method), 135 static method), 239
  - create\_cmd() (digi.xbee.models.filesystem.FileIdNameComdeate\_packet() (digi.xbee.packets.cellular.RXSMSPacket class method), 137 static method), 248
  - create\_cmd() (digi.xbee.models.filesystem.FSCmd create\_packet() (digi.xbee.packets.cellular.TXSMSPacket class method), 133 static method), 251

- create\_packet() (digi.xbee.packets.common.ATCommPackette\_packet() (digi.xbee.packets.raw.RX64IOPacket static method), 254 static method), 360
- create\_packet() (digi.xbee.packets.common.ATComm@uewePacketacket() (digi.xbee.packets.raw.RX64Packet static method), 257 static method), 353
- create\_packet() (digi.xbee.packets.common.ATCommResponsePacket\*() (digi.xbee.packets.raw.TX16Packet static method), 260 static method), 347
- create\_packet() (digi.xbee.packets.common.ExplicitAddressingPacketet() (digi.xbee.packets.raw.TX64Packet static method), 291 static method), 344
- create\_packet() (digi.xbee.packets.common.ExplicitRXhvelictaeorpackeet() (digi.xbee.packets.raw.TXStatusPacket static method), 297 static method), 350
- create\_packet() (digi.xbee.packets.common.IODataSampleRxIndicatoeRackdtdigi.xbee.packets.relay.UserDataRelayOutputPacke static method), 287 static method), 371
- create\_packet() (digi.xbee.packets.common.ModemStatusEdacket() (digi.xbee.packets.relay.UserDataRelayPacket static method), 283 static method), 368
- create\_packet() (digi.xbee.packets.common.ReceivePacketate\_packet() (digi.xbee.packets.socket.SocketBindListenPacket static method), 264 static method), 407
- create\_packet() (digi.xbee.packets.common.RemoteATGcmmandPacket() (digi.xbee.packets.socket.SocketClosePacket static method), 268 static method), 394
- create\_packet() (digi.xbee.packets.common.RemoteAECommon\_RemoteAEC
- create\_packet() (digi.xbee.packets.common.TransmitRackette\_packet() (digi.xbee.packets.socket.SocketConnectPacket static method), 276 static method), 388
- create\_packet() (digi.xbee.packets.common.TransmitStatusRacket() (digi.xbee.packets.socket.SocketConnectResponsePackets.tatic method), 280

  static method), 391
- create\_packet() (digi.xbee.packets.devicecloud.DeviceRequestPacketket() (digi.xbee.packets.socket.SocketCreatePacket static method), 300 static method), 374
- static method), 300 static method), 374
  create\_packet() (digi.xbee.packets.devicecloud.DeviceResponsePacket() (digi.xbee.packets.socketCreateResponsePacket())
- static method), 303

  create\_packet() (digi.xbee.packets.devicecloud.DeviceResponseStatusEncket(digi.xbee.packets.socket.SocketListenResponsePackets.static method), 306

  static method), 410
- create\_packet() (digi.xbee.packets.devicecloud.FrameErearPacket() (digi.xbee.packets.socket.SocketNewIPv4ClientPacket static method), 309

  static method), 413
- create\_packet() (digi.xbee.packets.devicecloud.SendDataRetque\_ptBacket() (digi.xbee.packets.socket.SocketOptionRequestPacket static method), 312 static method), 380
- create\_packet() (digi.xbee.packets.devicecloud.SendDataRetsponsePacket() (digi.xbee.packets.socket.SocketOptionResponsePackets static method), 317 static method), 384
- create\_packet() (digi.xbee.packets.digimesh.RouteInformationPacketset() (digi.xbee.packets.socket.SocketReceiveFromPacket static method), 319

  static method), 421

  create\_packet() (digi.xbee.packets filesystem FSRequestPacket, packet() (digi.xbee.packets.socket.SocketReceivePacket)
- $\label{lem:create_packet} \textit{create_packet}. () \ (\textit{digi.xbee.packets.filesystem.FSRequestRacket}\_\texttt{packet}\,() \ (\textit{digi.xbee.packets.socket.SocketReceivePackets.socket.socketReceivePackets.socket.socketReceivePackets.socket.socke$
- create\_packet() (digi.xbee.packets.filesystem.FSResponsePacket() (digi.xbee.packets.socket.SocketSendPacket static method), 326 static method), 400
- create\_packet() (digi.xbee.packets.filesystem.RemoteESRequestPacketet() (digi.xbee.packets.socket.SocketSendToPacket static method), 330 static method), 404
- create\_packet() (digi.xbee.packets.filesystem.RemoteESRespænsePacket() (digi.xbee.packets.socket.SocketStatePacket static method), 334
- create\_packet() (digi.xbee.packets.network.RXIPv4Packetate\_packet() (digi.xbee.packets.wifi.IODataSampleRxIndicatorWijstatic method), 337

  static method), 426
- create\_packet() (digi.xbee.packets.network.TXIPv4Packetate\_packet() (digi.xbee.packets.wifi.RemoteATCommandResponse static method), 340 static method), 435
- create\_packet() (digi.xbee.packets.raw.RX16IOPacketreate\_packet() (digi.xbee.packets.wifi.RemoteATCommandWifiPacketstatic method), 365

  static method), 430
- create\_packet() (digi.xbee.packets.raw.RX16Packet create\_packet() (digi.xbee.packets.zigbee.CreateSourceRoutePacket static method), 357 static method), 448

<pre>create_packet() (digi.xbee.packets.zigbee.OTAFirm</pre>	wareUpddieStdnusPnokels.filesystem.ReadFileCmdResponse attribute), 147
<pre>create_packet() (digi.xbee.packets.zigbee.RegisterD</pre>	evicaSta(disfaxket.models.filesystem.WriteFileCmdRequest attribute), 149
<pre>create_packet() (digi.xbee.packets.zigbee.RegisterJo static method), 437</pre>	oining Devial Buckete.models.message.ExplicitXBeeMessage attribute), 200
$\verb create_packet()  \textit{(digi.xbee.packets.zigbee.RouteRec} $	ordHidicatdifladkee.models.message.IPMessage attribute),
static method), 444	202
<pre>create_socket_info()     (digi.xbee.models.info.SocketInfo</pre>	data (digi.xbee.models.message.SMSMessage at- tribute), 202
method), 191	data(digi.xbee.models.message.UserDataRelayMessage
<pre>create_source_route()</pre>	attribute), 203
(digi.xbee.devices.ZigBeeDevice method), 616	data (digi.xbee.models.message.XBeeMessage at- tribute), 199
<pre>create_xbee_device()</pre>	data (digi.xbee.packets.cellular.RXSMSPacket at-
(digi.xbee.devices.CellularDevice class	tribute), 249
<pre>method), 675 create_xbee_device()</pre>	data (digi.xbee.packets.cellular.TXSMSPacket at- tribute), 253
(digi.xbee.devices.DigiMeshDevice class method), 550	data (digi.xbee.packets.network.RXIPv4Packet attribute), 338
<pre>create_xbee_device()     (digi.xbee.devices.DigiPointDevice</pre>	data (digi.xbee.packets.network.TXIPv4Packet attribute), 343
method), 581	data(digi.xbee.packets.relay.UserDataRelayOutputPacket
create_xbee_device()	attribute), 373
(digi.xbee.devices.IPDevice class method), 650	data (digi.xbee.packets.relay.UserDataRelayPacket at- tribute), 369
<pre>create_xbee_device()</pre>	DataReceived (class in digi.xbee.reader), 974
(digi.xbee.devices.LPWANDevice class method), 700	DEFAULT_TIME_BETWEEN_REQUESTS (digi.xbee.devices.XBeeNetwork attribute),
<pre>create_xbee_device()</pre>	874
(digi.xbee.devices.NBIoTDevice class method),	DEFAULT_TIME_BETWEEN_SCANS
725	(digi.xbee.devices.XBeeNetwork attribute),
<pre>create_xbee_device()</pre>	874
(digi.xbee.devices.Raw802Device class method), 518	<pre>del_bluetooth_data_received_callback()    (digi.xbee.devices.CellularDevice method), 676</pre>
create_xbee_device()	del_bluetooth_data_received_callback()
(digi.xbee.devices.WiFiDevice class method),	(digi.xbee.devices.DigiMeshDevice method), 550
<pre>create_xbee_device()</pre>	<pre>del_bluetooth_data_received_callback()</pre>
(digi.xbee.devices.XBeeDevice class method), 485	(digi.xbee.devices.DigiPointDevice method), 582
<pre>create_xbee_device()</pre>	<pre>del_bluetooth_data_received_callback()</pre>
(digi.xbee.devices.ZigBeeDevice class method),	(digi.xbee.devices.IPDevice method), 650
619	<pre>del_bluetooth_data_received_callback()</pre>
CreateDirCmdRequest (class in	(digi.xbee.devices.LPWANDevice method), 700
digi.xbee.models.filesystem), 156	del_bluetooth_data_received_callback()
CreateDirCmdResponse (class in	(digi.xbee.devices.NBIoTDevice method), 726
<pre>digi.xbee.models.filesystem), 158 CreateSourceRoutePacket (class in</pre>	<pre>del_bluetooth_data_received_callback()      (digi.xbee.devices.Raw802Device method), 518</pre>
CreateSourceRoutePacket (class in digi.xbee.packets.zigbee), 447	del_bluetooth_data_received_callback()
argunoccipacionistegocci, ++1	(digi.xbee.devices.WiFiDevice method), 758
D	del_bluetooth_data_received_callback()
daemon (digi.xbee.reader.PacketListener attribute), 993	(digi.xbee.devices.XBeeDevice method), 493
(	<pre>del_bluetooth_data_received_callback()</pre>

(digi.xbee.devices.ZigBeeDevice meth	od), 619	922	
del_bluetooth_data_received_callback() (digi.xbee.reader.PacketListener method), 998		<pre>del_discovery_process_finished_callback()       (digi.xbee.devices.Raw802Network method),</pre>	
del_data_received_callback()		899	
(digi.xbee.devices.CellularDevice 676	method),	<pre>del_discovery_process_finished_callback()      (digi.xbee.devices.XBeeNetwork method), 879</pre>	
del_data_received_callback()		<pre>del_discovery_process_finished_callback()</pre>	
(digi.xbee.devices.DigiMeshDevice 550	method),	(digi.xbee.devices.ZigBeeNetwork method),	
del_data_received_callback()		<pre>del_end_discovery_scan_callback()</pre>	
(digi.xbee.devices.DigiPointDevice 582	method),	(digi.xbee.devices.DigiMeshNetwork method), 911	
del_data_received_callback()		<pre>del_end_discovery_scan_callback()</pre>	
(digi.xbee.devices.IPDevice method),	647	(digi.xbee.devices.DigiPointNetwork method),	
del_data_received_callback()		922	
(digi.xbee.devices.LPWANDevice 700	method),	<pre>del_end_discovery_scan_callback()           (digi.xbee.devices.Raw802Network method),</pre>	
del_data_received_callback()		899	
(digi.xbee.devices.NBIoTDevice	method),	<pre>del_end_discovery_scan_callback()</pre>	
726		(digi.xbee.devices.XBeeNetwork method),	
del_data_received_callback()		879	
(digi.xbee.devices.Raw802Device 518	method),	<pre>del_end_discovery_scan_callback()      (digi.xbee.devices.ZigBeeNetwork method),</pre>	
del_data_received_callback()	D 750	888	
(digi.xbee.devices.WiFiDevice method	1), 738	del_expl_data_received_callback()	
del_data_received_callback() (digi.xbee.devices.XBeeDevice	method),	(digi.xbee.devices.CellularDevice method), 676	
492		del_expl_data_received_callback()	
del_data_received_callback() (digi.xbee.devices.ZigBeeDevice	method),	(digi.xbee.devices.DigiMeshDevice method), 550	
619		del_expl_data_received_callback()	
del_data_received_callback()	an oth od)	(digi.xbee.devices.DigiPointDevice method), 582	
(digi.xbee.reader.PacketListener 997	method),	del_expl_data_received_callback()	
del_device_discovered_callback()		(digi.xbee.devices.IPDevice method), 647	
(digi.xbee.devices.DigiMeshNetwork	method),	del_expl_data_received_callback()	
910	,	(digi.xbee.devices.LPWANDevice method),	
<pre>del_device_discovered_callback()</pre>		701	
(digi.xbee.devices.DigiPointNetwork	method),	<pre>del_expl_data_received_callback()</pre>	
922		(digi.xbee.devices.NBIoTDevice method),	
<pre>del_device_discovered_callback()</pre>		726	
(digi.xbee.devices.Raw802Network	method),	<pre>del_expl_data_received_callback()</pre>	
899		(digi.xbee.devices.Raw802Device method),	
del_device_discovered_callback()		518	
(digi.xbee.devices.XBeeNetwork	method),	<pre>del_expl_data_received_callback()</pre>	
878		(digi.xbee.devices.WiFiDevice method), 758	
<pre>del_device_discovered_callback()</pre>	.1 1	<pre>del_expl_data_received_callback()</pre>	
(digi.xbee.devices.ZigBeeNetwork 887	method),	(digi.xbee.devices.XBeeDevice method), 493	
		(del_expl_data_received_callback()	
(digi.xbee.devices.DigiMeshNetwork 910		(digi.xbee.devices.ZigBeeDevice method), 620	
		(Mel_explicit_data_received_callback()	
(digi.xbee.devices.DigiPointNetwork	method),	(digi.xbee.reader.PacketListener method), 997	

del_	_fs_frame_received_callback()     (digi.xbee.devices.CellularDevice method), 676	<pre>del_io_sample_received_callback()           (digi.xbee.devices.IPDevice method), 651 del_io_sample_received_callback()</pre>
del_	_fs_frame_received_callback()     (digi.xbee.devices.DigiMeshDevice method),	(digi.xbee.devices.LPWANDevice method), 701
del_	551 _fs_frame_received_callback()     (digi.xbee.devices.DigiPointDevice method),	<pre>del_io_sample_received_callback()      (digi.xbee.devices.NBIoTDevice method),      726</pre>
del_	582 _fs_frame_received_callback() (digi.xbee.devices.IPDevice method), 650	<pre>del_io_sample_received_callback()           (digi.xbee.devices.Raw802Device method),</pre>
del_	_fs_frame_received_callback()     (digi.xbee.devices.LPWANDevice method),	<pre>del_io_sample_received_callback()       (digi.xbee.devices.WiFiDevice method), 758</pre>
del_	701 _fs_frame_received_callback()     (digi.xbee.devices.NBIoTDevice method),	<pre>del_io_sample_received_callback()      (digi.xbee.devices.XBeeDevice method),      493</pre>
del_	726 _fs_frame_received_callback()     (digi.xbee.devices.Raw802Device method),	<pre>del_io_sample_received_callback()      (digi.xbee.devices.ZigBeeDevice method), 620</pre>
del_	518 _fs_frame_received_callback() (digi.xbee.devices.WiFiDevice method), 758	<pre>del_io_sample_received_callback()      (digi.xbee.reader.PacketListener method),      997</pre>
del_	_fs_frame_received_callback()    (digi.xbee.devices.XBeeDevice method), 493	<pre>del_ip_data_received_callback()           (digi.xbee.devices.CellularDevice method),</pre>
del_	_fs_frame_received_callback()     (digi.xbee.devices.ZigBeeDevice method), 620	<pre>del_ip_data_received_callback()</pre>
del_	_fs_frame_received_callback()     (digi.xbee.reader.PacketListener method),	(digi.xbee.devices.LPWANDevice method), 701
del_	998 _init_discovery_scan_callback()     (digi.xbee.devices.DigiMeshNetwork method),	<pre>del_ip_data_received_callback()       (digi.xbee.devices.NBIoTDevice method),       726</pre>
del_	911 _init_discovery_scan_callback() (disi_xhaa_dayiaas_Disi_PointNatusark_mathad)	<pre>del_ip_data_received_callback()           (digi.xbee.devices.WiFiDevice method), 758 del_ip_data_received_callback()</pre>
del_	(digi.xbee.devices.DigiPointNetwork method), 922 _init_discovery_scan_callback()	(digi.xbee.reader.PacketListener method), 997
	(digi.xbee.devices.Raw802Network method), 899	<pre>del_micropython_data_received_callback()</pre>
	_init_discovery_scan_callback()	<pre>del_micropython_data_received_callback()           (digi.xbee.devices.DigiMeshDevice method),</pre>
	(digi.xbee.devices.ZigBeeNetwork method), 888 _io_sample_received_callback()	<pre>del_micropython_data_received_callback()           (digi.xbee.devices.DigiPointDevice method),</pre>
aeı_	(digi.xbee.devices.CellularDevice method), 672	del_micropython_data_received_callback() (digi.xbee.devices.IPDevice method), 651
del_	_io_sample_received_callback()     (digi.xbee.devices.DigiMeshDevice method),     551	<pre>del_micropython_data_received_callback()</pre>
del_	_io_sample_received_callback()     (digi.xbee.devices.DigiPointDevice method),     582	(digi.xbee.devices.NBIoTDevice method), 726 del_micropython_data_received_callback() (digi.xbee.devices.Raw802Device method), 519

```
del_micropython_data_received_callback()
                                                         (digi.xbee.devices.DigiPointDevice
                                                                                        method),
        (digi.xbee.devices.WiFiDevice method), 758
                                                         582
del_micropython_data_received_callback() del_packet_received_callback()
        (digi.xbee.devices.XBeeDevice method), 493
                                                         (digi.xbee.devices.IPDevice method), 651
del_micropython_data_received_callback() del_packet_received_callback()
        (digi.xbee.devices.ZigBeeDevice method), 620
                                                         (digi.xbee.devices.LPWANDevice
                                                                                        method),
del micropython data received callback()
        (digi.xbee.reader.PacketListener method), 998
                                                del_packet_received_callback()
del_modem_status_received_callback()
                                                         (digi.xbee.devices.NBIoTDevice
                                                                                        method),
                                                         726
        (digi.xbee.devices.CellularDevice method), 676
del_modem_status_received_callback()
                                                del_packet_received_callback()
        (digi.xbee.devices.DigiMeshDevice
                                                         (digi.xbee.devices.Raw802Device
                                                                                        method),
                                                del_packet_received_callback()
del_modem_status_received_callback()
        (digi.xbee.devices.DigiPointDevice
                                                         (digi.xbee.devices.WiFiDevice method), 758
                                       method),
        582
                                                del_packet_received_callback()
                                                         (digi.xbee.devices.XBeeDevice
del_modem_status_received_callback()
                                                                                        method),
                                                         492
        (digi.xbee.devices.IPDevice method), 651
del_modem_status_received_callback()
                                                del_packet_received_callback()
        (digi.xbee.devices.LPWANDevice method), 701
                                                         (digi.xbee.devices.ZigBeeDevice
                                                                                        method),
del_modem_status_received_callback()
                                                         620
        (digi.xbee.devices.NBIoTDevice method), 726
                                                del_packet_received_callback()
del_modem_status_received_callback()
                                                         (digi.xbee.reader.PacketListener
                                                                                        method),
        (digi.xbee.devices.Raw802Device method), 519
                                                         997
del_modem_status_received_callback()
                                                del_packet_received_from_callback()
        (digi.xbee.devices.WiFiDevice method), 758
                                                         (digi.xbee.devices.DigiMeshNetwork method),
del_modem_status_received_callback()
        (digi.xbee.devices.XBeeDevice method), 493
                                                del_packet_received_from_callback()
del_modem_status_received_callback()
                                                         (digi.xbee.devices.DigiPointNetwork method),
        (digi.xbee.devices.ZigBeeDevice method), 620
del_modem_status_received_callback()
                                                del_packet_received_from_callback()
        (digi.xbee.reader.PacketListener method), 997
                                                         (digi.xbee.devices.Raw802Network
                                                                                        method),
del_network_modified_callback()
                                                         900
        (digi.xbee.devices.DigiMeshNetwork method),
                                                del_packet_received_from_callback()
                                                         (digi.xbee.devices.XBeeNetwork method), 879
del_network_modified_callback()
                                                del_packet_received_from_callback()
        (digi.xbee.devices.DigiPointNetwork method),
                                                         (digi.xbee.devices.ZigBeeNetwork
                                                                                        method),
del_network_modified_callback()
                                                del_packet_received_from_callback()
        (digi.xbee.devices.Raw802Network
                                       method),
                                                         (digi.xbee.reader.PacketListener method), 997
                                                del route info callback()
del_network_modified_callback()
                                                         (digi.xbee.reader.PacketListener
                                                                                        method),
        (digi.xbee.devices.XBeeNetwork
                                       method),
                                                del_route_received_callback()
del_network_modified_callback()
                                                         (digi.xbee.devices.CellularDevice
                                                                                        method),
                                                         677
        (digi.xbee.devices.ZigBeeNetwork
                                       method),
        888
                                                del_route_received_callback()
del_packet_received_callback()
                                                         (digi.xbee.devices.DigiMeshDevice
                                                                                        method),
        (digi.xbee.devices.CellularDevice
                                       method),
                                                del_route_received_callback()
del_packet_received_callback()
                                                         (digi.xbee.devices.DigiPointDevice
                                                                                        method),
        (digi.xbee.devices.DigiMeshDevice
                                       method),
        551
                                                del_route_received_callback()
del_packet_received_callback()
                                                         (digi.xbee.devices.IPDevice method), 651
```

<pre>del_route_received_callback()</pre>	(digi.xbee.reader.PacketListener method), 998
(digi.xbee.devices.LPWANDevice method),	<pre>del_socket_data_received_from_callback()           (digi.xbee.devices.CellularDevice method), 677</pre>
del_route_received_callback()	del_socket_data_received_from_callback()
(digi.xbee.devices.NBIoTDevice method),	(digi.xbee.devices.DigiMeshDevice method), 551
<pre>del_route_received_callback()      (digi.xbee.devices.Raw802Device method),      519</pre>	<pre>del_socket_data_received_from_callback()           (digi.xbee.devices.DigiPointDevice method),</pre>
<pre>del_route_received_callback()       (digi.xbee.devices.WiFiDevice method), 759</pre>	<pre>del_socket_data_received_from_callback()           (digi.xbee.devices.IPDevice method), 651</pre>
<pre>del_route_received_callback()</pre>	<pre>del_socket_data_received_from_callback()</pre>
(digi.xbee.devices.XBeeDevice method), 495	(digi.xbee.devices.LPWANDevice method), 701 del_socket_data_received_from_callback()
<pre>del_route_received_callback()</pre>	(digi.xbee.devices.NBIoTDevice method), 727
(digi.xbee.devices.ZigBeeDevice method),	<pre>del_socket_data_received_from_callback()</pre>
620	(digi.xbee.devices.Raw802Device method), 519
<pre>del_route_record_received_callback()</pre>	<pre>del_socket_data_received_from_callback()</pre>
(digi.xbee.reader.PacketListener method), 998	(digi.xbee.devices.WiFiDevice method), 759
del_sms_callback()	<pre>del_socket_data_received_from_callback()</pre>
(digi.xbee.devices.CellularDevice method),	(digi.xbee.devices.XBeeDevice method), 493
670	del_socket_data_received_from_callback()
<pre>del_sms_callback()       (digi.xbee.devices.LPWANDevice method),</pre>	(digi.xbee.devices.ZigBeeDevice method), 620 del_socket_data_received_from_callback()
697	(digi.xbee.reader.PacketListener method), 998
del_sms_callback()	del_socket_state_callback()
(digi.xbee.devices.NBIoTDevice method),	(digi.xbee.xsocket.socket method), 1014
727	<pre>del_socket_state_received_callback()</pre>
<pre>del_sms_received_callback()</pre>	(digi.xbee.devices.CellularDevice method), 677
(digi.xbee.reader.PacketListener method),	<pre>del_socket_state_received_callback()</pre>
997	(digi.xbee.devices.DigiMeshDevice method),
del_socket_data_received_callback()	551
(digi.xbee.devices.CellularDevice method), 677	del_socket_state_received_callback()
<pre>del_socket_data_received_callback()           (digi.xbee.devices.DigiMeshDevice method),</pre>	(digi.xbee.devices.DigiPointDevice method), 583
(digi.xbee.devices.DigiMeshDevice method), 551	del_socket_state_received_callback()
del_socket_data_received_callback()	(digi.xbee.devices.IPDevice method), 651
(digi.xbee.devices.DigiPointDevice method),	del_socket_state_received_callback()
583	(digi.xbee.devices.LPWANDevice method), 701
<pre>del_socket_data_received_callback()</pre>	<pre>del_socket_state_received_callback()</pre>
(digi.xbee.devices.IPDevice method), 651	(digi.xbee.devices.NBIoTDevice method), 727
<pre>del_socket_data_received_callback()     (digi.xbee.devices.LPWANDevice method), 701</pre>	<pre>del_socket_state_received_callback()       (digi.xbee.devices.Raw802Device method), 519</pre>
<pre>del_socket_data_received_callback()           (digi.xbee.devices.NBIoTDevice method), 727</pre>	<pre>del_socket_state_received_callback()       (digi.xbee.devices.WiFiDevice method), 759</pre>
del_socket_data_received_callback()	del_socket_state_received_callback()
(digi.xbee.devices.Raw802Device method), 519	(digi.xbee.devices.XBeeDevice method), 493
del_socket_data_received_callback()	del_socket_state_received_callback()
(digi.xbee.devices.WiFiDevice method), 759	(digi.xbee.devices.ZigBeeDevice method), 620
<pre>del_socket_data_received_callback()</pre>	<pre>del_socket_state_received_callback()</pre>
(digi.xbee.devices.XBeeDevice method), 493	(digi.xbee.reader.PacketListener method), 998
del_socket_data_received_callback()	del_user_data_relay_received_callback()
(digi.xbee.devices.ZigBeeDevice method), 620	(digi.xbee.devices.CellularDevice method), 677
<pre>del_socket_data_received_callback()</pre>	<pre>del_user_data_relay_received_callback()</pre>

(digi.xbee.devices.DigiMeshDevice method), 551	attribute), 208
del_user_data_relay_received_callback()	description (digi.xbee.models.options.RegisterKeyOptions attribute), 209
(digi.xbee.devices.DigiPointDevice method), 583	description (digi.xbee.models.options.SendDataRequestOptions attribute), 207
del_user_data_relay_received_callback() (digi.xbee.devices.IPDevice method), 651	description (digi.xbee.models.options.SocketOption attribute), 210
<pre>del_user_data_relay_received_callback()      (digi.xbee.devices.LPWANDevice method), 701</pre>	description (digi.xbee.models.options.XBeeLocalInterface attribute), 209
<pre>del_user_data_relay_received_callback()      (digi.xbee.devices.NBIoTDevice method), 727</pre>	description (digi.xbee.models.protocol.IPProtocol attribute), 213
<pre>del_user_data_relay_received_callback()      (digi.xbee.devices.Raw802Device method), 519</pre>	description (digi.xbee.models.protocol.Role attribute), 213
<pre>del_user_data_relay_received_callback()       (digi.xbee.devices.WiFiDevice method), 759</pre>	description (digi.xbee.models.protocol.XBeeProtocol attribute), 212
<pre>del_user_data_relay_received_callback()       (digi.xbee.devices.XBeeDevice method), 493</pre>	description (digi.xbee.models.status.AssociationIndicationStatus attribute), 220
del_user_data_relay_received_callback() (digi.xbee.devices.ZigBeeDevice method), 620	description (digi.xbee.models.status.ATCommandStatus attribute), 214
<pre>del_user_data_relay_received_callback()</pre>	description (digi.xbee.models.status.CellularAssociationIndicationState attribute), 220
digi.xbee.models.filesystem), 177	description (digi.xbee.models.status.DeviceCloudStatus attribute), 221
digi.xbee.models.filesystem), 179	description (digi.xbee.models.status.DiscoveryStatus attribute), 214
deprecated() (in module digi.xbee.util.utils), 461 depth (digi.xbee.models.zdo.Neighbor attribute), 235	description (digi.xbee.models.status.EmberBootloaderMessageType attribute), 224
desc_capabilities	description (digi.xbee.models.status.FrameError at-
(digi.xbee.models.zdo.NodeDescriptor at- tribute), 229	tribute), 222 description (digi.xbee.models.status.FSCommandStatus
description (digi.xbee.devices.NetworkEventReason attribute), 931	attribute), 227 description (digi.xbee.models.status.ModemStatus
<pre>description (digi.xbee.devices.NetworkEventType</pre>	attribute), 218 description (digi.xbee.models.status.NetworkDiscoveryStatus
description (digi.xbee.io.IOLine attribute), 957 description (digi.xbee.models.accesspoint.WiFiEncryp	
attribute), 126 description(digi.xbee.models.atcomm.ATStringComm	tribute), 218 namedscription (digi.xbee.models.status.SocketInfoState
attribute), 129 description (digi.xbee.models.filesystem.FSCmdType	attribute), 226 description (digi.xbee.models.status.SocketState at-
attribute), 131 description (digi.xbee.models.hw.HardwareVersion	tribute), 225 description (digi.xbee.models.status.SocketStatus
<pre>attribute), 189 description(digi.xbee.models.mode.APIOutputMode</pre>	attribute), 225 description (digi.xbee.models.status.TransmitStatus
attribute), 193 description (digi.xbee.models.mode.APIOutputModeB	attribute), 216  Bitlescription (digi.xbee.models.status.WiFiAssociationIndicationStatus
attribute), 193	attribute), 222  ledescription (digi.xbee.models.status.ZigbeeRegisterStatus
attribute), 194 description (digi.xbee.models.mode.NeighborDiscove	attribute), 223
attribute), 195	tribute), 238
description (digi.xbee.models.mode.OperatingMode attribute), 192	
description (digi.xbee.models.options.DiscoveryOptio	

```
969
                                                    determine protocol()
description (digi.xbee.profile.XBeeSettingFormat at-
                                                             (digi.xbee.devices.IPDevice method), 651
        tribute), 966
                                                    determine protocol()
description (digi.xbee.profile.XBeeSettingType at-
                                                             (digi.xbee.devices.LPWANDevice
                                                                                               method),
        tribute), 965
dest address(digi.xbee.packets.network.TXIPv4Packetdetermine protocol()
                                                             (digi.xbee.devices.NBIoTDevice
        attribute), 341
                                                                                               method),
dest_address(digi.xbee.packets.socket.SocketConnectPacket
                                                             727
        attribute), 389
                                                    determine_protocol()
dest_address(digi.xbee.packets.socket.SocketSendToPacket
                                                             (digi.xbee.devices.Raw802Device
                                                                                               method),
        attribute), 404
dest_address(digi.xbee.packets.wifi.RemoteATCommandWifePackete_protocol()
                                                             (digi.xbee.devices.RemoteDigiMeshDevice
        attribute), 431
DEST_ADDRESS_BINARY
                                                             method), 818
        (digi.xbee.packets.socket.SocketConnectPacket
                                                    determine_protocol()
        attribute), 388
                                                             (digi.xbee.devices.RemoteDigiPointDevice
DEST_ADDRESS_STRING
                                                             method), 836
        (digi.xbee.packets.socket.SocketConnectPacket
                                                    determine protocol()
                                                             (digi.xbee.devices.RemoteRaw802Device
        attribute), 388
dest_address_type
                                                             method), 800
        (digi.xbee.packets.socket.SocketConnectPacket
                                                    determine_protocol()
                                                             (digi.xbee.devices.RemoteXBeeDevice
        attribute), 389
dest_endpoint (digi.xbee.models.message.ExplicitXBeeMessage method), 783
        attribute), 200
                                                    determine protocol()
dest_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketigi.xbee.devices.RemoteZigBeeDevice
        attribute), 294
                                                             method), 857
dest_endpoint (digi.xbee.packets.common.ExplicitRXIndlictatcmRiscket_protocol()
                                                             (digi.xbee.devices.WiFiDevice method), 759
        attribute), 299
dest_interface(digi.xbee.packets.relay.UserDataRelayPacketmine_protocol()
        attribute), 369
                                                             (digi.xbee.devices.XBeeDevice
                                                                                               method),
dest_port (digi.xbee.models.message.IPMessage at-
                                                             497
        tribute), 201
                                                    determine_protocol()
dest_port (digi.xbee.packets.network.RXIPv4Packet
                                                             (digi.xbee.devices.ZigBeeDevice
                                                                                               method),
        attribute), 337
dest_port (digi.xbee.packets.network.TXIPv4Packet
                                                    DeviceCloudStatus
                                                                                     (class
                                                                                                     in
                                                             digi.xbee.models.status), 221
        attribute), 342
dest port (digi.xbee.packets.socket.SocketConnectPackeDeviceDiscovered (class in digi.xbee.reader), 978
        attribute), 389
                                                    DeviceRequestPacket
                                                                                       (class
                                                                                                     in
dest_port (digi.xbee.packets.socket.SocketSendToPacket
                                                             digi.xbee.packets.devicecloud), 299
        attribute), 405
                                                    DeviceResponsePacket
                                                                                       (class
                                                                                                     in
destination (digi.xbee.models.zdo.Route attribute),
                                                             digi.xbee.packets.devicecloud), 303
        231
                                                    DeviceResponseStatusPacket
                                                                                                     in
determine_protocol()
                                                             digi.xbee.packets.devicecloud), 306
        (digi.xbee.devices.AbstractXBeeDevice
                                                    DictKeys (class in digi.xbee.packets.base), 238
        method), 470
                                                    digi (module), 124
determine_protocol()
                                                    digi.xbee (module), 124
        (digi.xbee.devices.CellularDevice
                                          method),
                                                    digi.xbee.comm_interface(module), 464
        677
                                                    digi.xbee.devices (module), 467
determine_protocol()
                                                    digi.xbee.exception (module), 934
        (digi.xbee.devices.DigiMeshDevice
                                          method),
                                                    digi.xbee.filesystem (module), 937
                                                    digi.xbee.firmware (module), 954
        551
determine protocol()
                                                    digi.xbee.io (module), 957
        (digi.xbee.devices.DigiPointDevice
                                          method),
                                                    digi.xbee.models (module), 124
        583
                                                    digi.xbee.models.accesspoint (module), 124
```

```
digi.xbee.models.address (module), 195
                                                   digital_values (digi.xbee.io.IOSample attribute),
digi.xbee.models.atcomm (module), 126
                                                            960
digi.xbee.models.filesystem (module), 131
                                                   direction (digi.xbee.models.filesystem.CloseDirCmdRequest
digi.xbee.models.hw (module), 187
                                                            attribute), 163
digi.xbee.models.info (module), 190
                                                   direction (digi.xbee.models.filesystem.CloseDirCmdResponse
digi.xbee.models.message (module), 199
                                                            attribute), 165
digi.xbee.models.mode (module), 192
                                                   direction (digi.xbee.models.filesystem.CloseFileCmdRequest
digi.xbee.models.options (module), 203
                                                            attribute), 142
digi.xbee.models.protocol(module),211
                                                   direction (digi.xbee.models.filesystem.CloseFileCmdResponse
digi.xbee.models.status (module), 213
                                                            attribute), 144
digi.xbee.models.zdo(module), 227
                                                   {\tt direction} \ (digi.x bee.models. file system. Create Dir Cmd Request
digi.xbee.packets (module), 236
                                                            attribute), 157
digi.xbee.packets.aft (module), 236
                                                   direction (digi.xbee.models.filesystem.CreateDirCmdResponse
digi.xbee.packets.base (module), 238
                                                            attribute), 158
digi.xbee.packets.cellular (module), 247
                                                   direction (digi.xbee.models.filesystem.DeleteCmdRequest
digi.xbee.packets.common (module), 253
                                                            attribute), 178
                                                   direction (digi.xbee.models.filesystem.DeleteCmdResponse
digi.xbee.packets.devicecloud
                                         (module),
        299
                                                            attribute), 179
digi.xbee.packets.digimesh (module), 318
                                                   direction (digi.xbee.models.filesystem.FileIdCmd at-
digi.xbee.packets.factory (module), 455
                                                            tribute), 135
digi.xbee.packets.filesystem (module), 323
                                                   direction (digi.xbee.models.filesystem.FileIdNameCmd
digi.xbee.packets.network (module), 336
                                                            attribute), 137
digi.xbee.packets.raw (module), 343
                                                                      (digi.xbee.models.filesystem.FSCmd
                                                   direction
digi.xbee.packets.relay (module), 367
                                                            attribute), 132
digi.xbee.packets.socket (module), 374
                                                   direction (digi.xbee.models.filesystem.GetPathIdCmdRequest
digi.xbee.packets.wifi (module), 425
                                                            attribute), 171
digi.xbee.packets.zigbee (module), 437
                                                   direction(digi.xbee.models.filesystem.GetPathIdCmdResponse
digi.xbee.profile (module), 963
                                                            attribute), 173
digi.xbee.reader (module), 971
                                                   direction (digi.xbee.models.filesystem.HashFileCmdRequest
digi.xbee.recovery (module), 1004
                                                            attribute), 153
digi.xbee.sender (module), 1004
                                                   direction (digi.xbee.models.filesystem.HashFileCmdResponse
digi.xbee.serial (module), 1006
                                                            attribute), 155
digi.xbee.util (module), 457
                                                   direction (digi.xbee.models.filesystem.OpenDirCmdRequest
digi.xbee.util.utils(module), 457
                                                            attribute), 160
digi.xbee.util.xmodem (module), 461
                                                   direction (digi.xbee.models.filesystem.OpenDirCmdResponse
digi.xbee.xsocket (module), 1010
                                                            attribute), 162
DIGIMESH MODE (digi.xbee.models.options.ReceiveOptionsirection (digi.xbee.models.filesystem.OpenFileCmdRequest
        attribute), 204
                                                            attribute), 139
\verb|DIGIMESH_MODE| (digi.xbee.models.options.TransmitOptionisrection (digi.xbee.models.filesystem.OpenFileCmdResponse)|
        attribute), 206
                                                            attribute), 141
DigiMeshDevice (class in digi.xbee.devices), 541
                                                   direction (digi.xbee.models.filesystem.ReadDirCmdRequest
DigiMeshNetwork (class in digi.xbee.devices), 908
                                                            attribute), 167
DigiPointDevice (class in digi.xbee.devices), 573
                                                   direction (digi.xbee.models.filesystem.ReadDirCmdResponse
DigiPointNetwork (class in digi.xbee.devices), 919
                                                            attribute), 168
                       (digi.xbee.io.IOSample
digital_hsb_mask
                                                   direction (digi.xbee.models.filesystem.ReadFileCmdRequest
        tribute), 960
                                                            attribute), 146
DIGITAL_IN (digi.xbee.io.IOMode attribute), 962
                                                   direction (digi.xbee.models.filesystem.ReadFileCmdResponse
                       (digi.xbee.io.IOSample
digital_lsb_mask
                                               at-
                                                            attribute), 148
        tribute), 960
                                                   direction (digi.xbee.models.filesystem.RenameCmdRequest
digital_mask (digi.xbee.io.IOSample attribute), 960
                                                            attribute), 174
DIGITAL_OUT_HIGH (digi.xbee.io.IOMode attribute),
                                                   {\tt direction} ({\it digi.xbee.models.filesystem.RenameCmdResponse}
                                                            attribute), 176
DIGITAL_OUT_LOW (digi.xbee.io.IOMode attribute),
                                                   direction (digi.xbee.models.filesystem.UnknownFSCmd
        962
                                                            attribute), 134
```

```
direction (digi.xbee.models.filesystem.VolFormatCmdRequest
                                                              (digi.xbee.devices.RemoteZigBeeDevice
        attribute), 184
                                                             method), 857
direction (digi.xbee.models.filesystem.VolFormatCmdResponse) le bluetooth()
                                                              (digi.xbee.devices.WiFiDevice method), 759
        attribute), 186
direction(digi.xbee.models.filesystem.VolStatCmdRequest sable_bluetooth()
        attribute), 181
                                                             (digi.xbee.devices.XBeeDevice
                                                                                               method),
direction (digi.xbee.models.filesystem.VolStatCmdResponse
                                                              497
        attribute), 183
                                                     disable_bluetooth()
direction (digi.xbee.models.filesystem.WriteFileCmdRequest
                                                              (digi.xbee.devices.ZigBeeDevice
                                                                                               method),
        attribute), 150
direction (digi.xbee.models.filesystem.WriteFileCmdResphorsseble_logger() (in module digi.xbee.util.utils),
        attribute), 151
                                                              461
DirResponseFlag
                               (class
                                                 in DISABLE_RETRIES_AND_REPAIR
        digi.xbee.models.options), 211
                                                             (digi.xbee.models.options.TransmitOptions
{\tt DISABLE\_ACK}\ (digi.xbee.models.options. Remote ATC mdOptions
                                                             attribute), 205
        attribute), 207
                                                     DISABLED (digi.xbee.io.IOMode attribute), 962
DISABLE_ACK(digi.xbee.models.options.TransmitOptionsdisconnect()
                                                                             (digi.xbee.devices.WiFiDevice
        attribute), 205
                                                             method), 751
disable_bluetooth()
                                                    disconnect () (digi.xbee.filesystem.LocalXBeeFileSystemManager
        (digi.xbee.devices.AbstractXBeeDevice
                                                             method), 951
        method), 482
                                                    discover_device()
disable bluetooth()
                                                             (digi.xbee.devices.DigiMeshNetwork method),
                                                             911
        (digi.xbee.devices.CellularDevice
                                          method),
        677
                                                    discover device()
                                                             (digi.xbee.devices.DigiPointNetwork method),
disable bluetooth()
        (digi.xbee.devices.DigiMeshDevice
                                          method),
                                                    discover_device()
                                                              (digi.xbee.devices.Raw802Network
disable_bluetooth()
                                                                                               method),
        (digi.xbee.devices.DigiPointDevice
                                                              900
                                          method),
                                                    discover_device()
disable_bluetooth() (digi.xbee.devices.IPDevice
                                                             (digi.xbee.devices.XBeeNetwork
                                                                                               method),
        method), 652
                                                              875
disable_bluetooth()
                                                    discover_device()
        (digi.xbee.devices.LPWANDevice
                                                             (digi.xbee.devices.ZigBeeNetwork
                                          method),
                                                                                               method),
        702
                                                              889
disable bluetooth()
                                                    discover devices()
        (digi.xbee.devices.NBIoTDevice
                                          method),
                                                             (digi.xbee.devices.DigiMeshNetwork method),
                                                              912
disable_bluetooth()
                                                    discover_devices()
                                                             (digi.xbee.devices.DigiPointNetwork method),
        (digi.xbee.devices.Raw802Device
                                          method),
        519
disable bluetooth()
                                                    discover devices()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                              (digi.xbee.devices.Raw802Network
                                                                                               method),
        method), 819
                                                              900
disable_bluetooth()
                                                     discover_devices()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                              (digi.xbee.devices.XBeeNetwork
                                                                                               method),
                                                              875
        method), 837
disable_bluetooth()
                                                     discover_devices()
        (digi.xbee.devices.RemoteRaw802Device
                                                             (digi.xbee.devices.ZigBeeNetwork
                                                                                               method),
        method), 800
disable_bluetooth()
                                                    {\tt DISCOVER\_MYSELF} \ (digi.x bee. models. options. Discovery Options
        (digi.xbee.devices.RemoteXBeeDevice
                                                             attribute), 208
        method), 783
                                                    discovery_status(digi.xbee.packets.common.TransmitStatusPacket
                                                              attribute), 281
disable_bluetooth()
```

	eryOptions digi.xbee.models.options)	(class , 207	in	enable_	_apply_changes() (digi.xbee.devices.WiFiDevice metho	d), 759
Discove	eryProcessFinished digi.xbee.reader), 978		s in	enable_	_apply_changes() (digi.xbee.devices.XBeeDevice	method),
	eryStatus ( <i>class in dig</i>	;i.xbee.mode	ls.status),		497	,,,
	214			enable_	_apply_changes()	
	ım () (in module digi.xbee				(digi.xbee.devices.ZigBeeDevice	method),
	TTEMPT_RD (digi.xbee.mo	odels.option:	s.Transmit <b>C</b>	_	621	
	attribute), 205 dr (digi.xbee.packets.digin	mash Poutali	oformation.		_APS_ENCRYPTION (digi.xbee.models.options.TransmitO	ntions
	attribute), 322	nesn.Routen	gormanom		attribute), 206	puons
Е				enable.	_bluetooth()	
	+ ] l M	(-1-			(digi.xbee.devices.AbstractXBeeDevi method), 482	ce
	ootloaderMessageTy		iss in	onahlo	_bluetooth()	
	digi.xbee.models.status), 2 (digi.xbee.reader.XBeeQ		A 1002	CHADIC.		method),
	_apply_changes()	иене тетос	<i>t)</i> , 1002		678	memou),
	_appry_changes () (digi.xbee.devices.Abstrac	ctXReeDevic	0	enable	_bluetooth()	
	method), 473	ADEEDEVIC	c		(digi.xbee.devices.DigiMeshDevice 552	method),
	_apply_changes() ( <i>digi.xbee.devices.Cellula</i>	ırDavica	method),	enable	_bluetooth()	
	677	TDevice	meinoa),	onas ro	(digi.xbee.devices.DigiPointDevice 583	method),
	_apply_changes() (digi.xbee.devices.DigiMe	eshDevice	method),	enable	_bluetooth() (digi.xbee.device.	s.IPDevice
	552	SILDEVICE	memou),		method), 652	
	_apply_changes()			enable_	_bluetooth()	
	(digi.xbee.devices.DigiPol 583	intDevice	method),		(digi.xbee.devices.LPWANDevice 702	method),
enable_	_apply_changes()			enable_	_bluetooth()	
	(digi.xbee.devices.IPDevi	ce method),	652		(digi.xbee.devices.NBIoTDevice	method),
	_apply_changes()				728	
	(digi.xbee.devices.LPWAN	<i>VDevice</i>	method),	enable_	_bluetooth()	.7 1
	702				(digi.xbee.devices.Raw802Device	method),
	_apply_changes()	ъ :	.1 1	onablo	520	
	(digi.xbee.devices.NBIoTI 727	Device	method),	enabre.	_bluetooth() (digi.xbee.devices.RemoteDigiMeshL	Device
	_apply_changes()			1 7	method), 819	
	(digi.xbee.devices.Raw802	2Device	method),	enable.		Davisa
1 7	520				(digi.xbee.devices.RemoteDigiPointL method), 837	revice
	_apply_changes()	DiaiMashD	aviaa	enahle	_bluetooth()	
	(digi.xbee.devices.Remote method), 819	DigimeshDi	evice	CHADIC.		vice
	_apply_changes()				method), 800	,,,,,
	_dppry_changes() (digi.xbee.devices.Remote	DigiPointDe	evice	enable	_bluetooth()	
	method), 837	8	.,,,,		(digi.xbee.devices.RemoteXBeeDevic	re
	_apply_changes()				<i>method</i> ), 784	
	(digi.xbee.devices.Remote	Raw802Dev	rice	enable_	_bluetooth()	
	method), 800				(digi.xbee.devices.RemoteZigBeeDev	rice
enable_	_apply_changes()				method), 857	
	(digi.xbee.devices.Remote	XBeeDevice	?	enable_	_bluetooth()	D 760
	method), 784			a 1- 7	(digi.xbee.devices.WiFiDevice metho	a), 760
	_apply_changes()	7' D D '		enable_	_bluetooth()	method),
	(digi.xbee.devices.Remote method), 857	zigBeeDevi	ce		(digi.xbee.devices.XBeeDevice 497	тетоа),

enable_bluetooth()	728
(digi.xbee.devices.ZigBeeDevice method), 621	execute_command() (digi.xbee.devices.Raw802Device method)
<pre>enable_logger() (in module digi.xbee.util.utils),      461</pre>	520 execute_command()
ENABLE_MULTICAST (digi.xbee.models.options.Transmi attribute), 206	
ENABLE_TRACE_ROUTE (digi.xbee.models.options.TransmitOptions attribute), 206	<pre>execute_command()      (digi.xbee.devices.RemoteDigiPointDevice      method), 837</pre>
ENABLE_UNICAST_NACK (digi.xbee.models.options.TransmitOptions attribute), 205	<pre>execute_command()      (digi.xbee.devices.RemoteRaw802Device      method), 801</pre>
ENABLE_UNICAST_TRACE_ROUTE	<pre>execute_command()</pre>
(digi.xbee.models.options.TransmitOptions attribute), 205	(digi.xbee.devices.RemoteXBeeDevice method), 784
encryption_type (digi.xbee.models.accesspoint.Access attribute), 125	ssPaincute_command()
EndDiscoveryScan (class in digi.xbee.reader), 991	method), 857
enter_at_command_mode() (in module digi.xbee.recovery), 1004	execute_command() (digi.xbee.devices.WiFiDevices method), 760
error (digi.xbee.models.zdo.NeighborFinder attribute), 235	execute_command() (digi.xbee.devices.XBeeDevice method), 497
error (digi.xbee.models.zdo.NeighborTableReader at-	<pre>execute_command()</pre>
tribute), 233	(digi.xbee.devices.ZigBeeDevice method).
error (digi.xbee.models.zdo.NodeDescriptorReader attribute), 227	621 ExplicitAddressingPacket (class in
error (digi.xbee.models.zdo.RouteTableReader attribute), 230	digi.xbee.packets.common), 290 ExplicitDataReceived (class in digi.xbee.reader).
error (digi.xbee.packets.devicecloud.FrameErrorPacket	979
attribute), 310	ExplicitRXIndicatorPacket (class in
EXCLUSIVE (digi.xbee.models.options.FileOpenRequestO	
attribute), 210	ExplicitXBeeMessage (class in
exec_at_cmd() (digi.xbee.firmware.UpdateConfigurer	digi.xbee.models.message), 199
static method), 955	export() (digi.xbee.devices.DigiMeshNetwork
execute_command()	method), 912
(digi.xbee.devices.AbstractXBeeDevice method), 468	export () (digi.xbee.devices.DigiPointNetwork method), 924
<pre>execute_command()      (digi.xbee.devices.CellularDevice method),</pre>	export () (digi.xbee.devices.Raw802Network method).
678	export() (digi.xbee.devices.XBeeNetwork method). 876
<pre>execute_command()      (digi.xbee.devices.DigiMeshDevice method),      552</pre>	export () (digi.xbee.devices.ZigBeeNetwork method). 889
<pre>execute_command()      (digi.xbee.devices.DigiPointDevice method),</pre>	extend() (digi.xbee.reader.BluetoothDataReceived method), 983
584	<pre>extend() (digi.xbee.reader.DataReceived method)</pre>
execute_command() (digi.xbee.devices.IPDevice method), 652	974 extend() (digi.xbee.reader.DeviceDiscovered method).
execute_command()	978
(digi.xbee.devices.LPWANDevice method), 702	extend() (digi.xbee.reader.DiscoveryProcessFinished method), 979
execute_command()	extend() (digi.xbee.reader.EndDiscoveryScan
(digi.xbee.devices.NBIoTDevice method),	method), 991

<pre>extend() (digi.xbee.reader.ExplicitDataReceived</pre>	FileSystemException, 938
method), 980	FileSystemFrameReceived (class in
<pre>extend() (digi.xbee.reader.FileSystemFrameReceived</pre>	digi.xbee.reader), 991
method), 992	FileSystemManager (class in digi.xbee.filesystem),
extend() (digi.xbee.reader.InitDiscoveryScan	939
method), 990	FileSystemNotSupportedException, 938
extend() (digi.xbee.reader.IOSampleReceived	firmware_description_file
method), 976	(digi.xbee.profile.XBeeProfile attribute),
extend() (digi.xbee.reader.IPDataReceived method),	970
981	firmware_version (digi.xbee.profile.XBeeProfile at-
extend() (digi.xbee.reader.MicroPythonDataReceived	tribute), 970
method), 984	FirmwareBaudrate (class in digi.xbee.profile), 963
extend() (digi.xbee.reader.ModemStatusReceived	FirmwareParity (class in digi.xbee.profile), 963
method), 975	FirmwareStopbits (class in digi.xbee.profile), 964
extend() (digi.xbee.reader.NetworkModified method), 977	FirmwareUpdateException, 936
extend() (digi.xbee.reader.PacketReceived method),	flags (digi.xbee.packets.devicecloud.DeviceRequestPacket attribute), 301
973	flash_firmware_option
extend() (digi.xbee.reader.PacketReceivedFrom	(digi.xbee.profile.XBeeProfile attribute),
method), 973	968
extend() (digi.xbee.reader.RelayDataReceived	FlashFirmwareOption (class in digi.xbee.profile),
method), 982	964
extend() (digi.xbee.reader.RouteInformationReceived	FLOOD (digi.xbee.models.mode.NeighborDiscoveryMode
method), 988	attribute), 194
extend() (digi.xbee.reader.RouteReceived method),	FlowControl (class in digi.xbee.serial), 1006
989	flush() (digi.xbee.reader.XBeeQueue method), 1003
<pre>extend() (digi.xbee.reader.RouteRecordIndicatorReceiv</pre>	
method), 987	method), 678
extend() (digi.xbee.reader.SMSReceived method), 981	flush_queues() (digi.xbee.devices.DigiMeshDevice
extend() (digi.xbee.reader.SocketDataReceived	method), 553
method), 986	<pre>flush_queues() (digi.xbee.devices.DigiPointDevice</pre>
<pre>extend() (digi.xbee.reader.SocketDataReceivedFrom</pre>	method), 584
method), 987	flush_queues() (digi.xbee.devices.IPDevice
extend() (digi.xbee.reader.SocketStateReceived	method), 653
method), 985	flush_queues() (digi.xbee.devices.LPWANDevice
extend() (digi.xbee.reader.XBeeEvent method), 972	method), 703
EXTENDED_TIMEOUT (digi.xbee.models.options.RemoteA	
attribute), 207	method), 728
F	flush_queues() (digi.xbee.devices.Raw802Device
	method), 521
$\verb file_data  (digi.x bee. packets. device cloud. Send Data Req$	ut st Packet queues () (digi.xbee.devices.WiFiDevice
attribute), 313	method), 760
$\verb file_hash  (digi.xbee.models.filesystem.HashFileCmdRef)   $	sponson_queues() (digi.xbee.devices.XBeeDevice
attribute), 155	method), 490
file_system_path (digi.xbee.profile.XBeeProfile at-	flush_queues() (digi.xbee.devices.ZigBeeDevice method), 622
tribute), 970	force_disassociate()
FileIdCmd (class in digi.xbee.models.filesystem), 135	(digi.xbee.devices.RemoteZigBeeDevice
FileIdNameCmd (class in digi.xbee.models.filesystem),	method), 854
136 FileOpenPaguage Option (class in	force_disassociate()
FileOpenRequestOption (class in	(digi.xbee.devices.ZigBeeDevice method),
digi.xbee.models.options), 210 FileProcess (class in digi.xbee.filesystem), 938	606
FileSystemElement (class in digi.xbee.filesystem), 938	format (digi.xbee.profile.XBeeProfileSetting attribute),
937	967

format() (digi.xbee.filesystem.FileSystemManager attribute), 324
method), 944 frame_id(digi.xbee.packets.filesystem.FSResponsePacket
<pre>format_filesystem()</pre>
(digi.xbee.filesystem.LocalXBeeFileSystemManag&rame_id (digi.xbee.packets.filesystem.RemoteFSRequestPacket method), 953 attribute), 331
frame_id (digi.xbee.packets.base.GenericXBeePacket frame_id(digi.xbee.packets.filesystem.RemoteFSResponsePacket attribute), 243 attribute), 333
frame_id(digi.xbee.packets.base.UnknownXBeePacket frame_id(digi.xbee.packets.network.RXIPv4Packet at- attribute), 346 tribute), 338
frame_id (digi.xbee.packets.base.XBeeAPIPacket at- tribute), 241 frame_id (digi.xbee.packets.network.TXIPv4Packet at- tribute), 341
frame_id (digi.xbee.packets.cellular.RXSMSPacket at- tribute), 249 frame_id (digi.xbee.packets.raw.RX16IOPacket attribute), 364
frame_id (digi.xbee.packets.cellular.TXSMSPacket at- tribute), 251 tribute), 358
frame_id (digi.xbee.packets.common.ATCommPacket frame_id (digi.xbee.packets.raw.RX64IOPacket attribute), 255 attribute), 362
frame_id (digi.xbee.packets.common.ATCommQueuePacketame_id (digi.xbee.packets.raw.RX64Packet at-attribute), 258 tribute), 355
frame_id (digi.xbee.packets.common.ATCommResponsePacketne_id (digi.xbee.packets.raw.TX16Packet at-attribute), 262 tribute), 348
frame_id (digi.xbee.packets.common.ExplicitAddressingPackete_id (digi.xbee.packets.raw.TX64Packet at-attribute), 345
frame_id (digi.xbee.packets.common.ExplicitRXIndicatorPackate_id (digi.xbee.packets.raw.TXStatusPacket at- attribute), 351
$\verb frame_id  (\textit{digi.xbee.packets.common.IODataSampleRxIndicatoePackets.relay.UserDataRelayOutputPackets)    \textit{digi.xbee.packets.relay.UserDataRelayOutputPackets}    \textit{digi.xbee.packets.relay.UserDataRelayOutputPackets}  $
attribute), 288 attribute), 372 frame_id(digi.xbee.packets.common.ModemStatusPacketframe_id(digi.xbee.packets.relay.UserDataRelayPacket
attribute), 284 attribute), 369
frame_id (digi.xbee.packets.common.ReceivePacket frame_id (digi.xbee.packets.socket.SocketBindListenPacket attribute), 265 attribute), 408
frame_id (digi.xbee.packets.common.RemoteATCommandPacket_id (digi.xbee.packets.socket.SocketClosePacket attribute), 269 attribute), 395
frame_id (digi.xbee.packets.common.RemoteATCommandResponsePackets.socket.SocketCloseResponsePacket attribute), 273  attribute), 398
frame_id (digi.xbee.packets.common.TransmitPacket frame_id (digi.xbee.packets.socket.SocketConnectPacket attribute), 278 attribute), 389
frame_id(digi.xbee.packets.common.TransmitStatusPacketrame_id(digi.xbee.packets.socket.SocketConnectResponsePacket attribute), 281 attribute), 392
frame_id(digi.xbee.packets.devicecloud.DeviceRequestPacketne_id(digi.xbee.packets.socket.SocketCreatePacket attribute), 301 attribute), 375
frame_id(digi.xbee.packets.devicecloud.DeviceResponsePackete_id(digi.xbee.packets.socket.SocketCreateResponsePacket attribute), 304 attribute), 378
frame_id(digi.xbee.packets.devicecloud.DeviceResponseStatusPacket(digi.xbee.packets.socket.SocketListenResponsePacket attribute), 307 attribute), 411
frame_id(digi.xbee.packets.devicecloud.FrameErrorPackerame_id(digi.xbee.packets.socket.SocketNewIPv4ClientPacket attribute), 310 attribute), 414
frame_id(digi.xbee.packets.devicecloud.SendDataRequestPacket_id(digi.xbee.packets.socket.SocketOptionRequestPacket attribute), 313 attribute), 382
frame_id(digi.xbee.packets.devicecloud.SendDataResportsePacketid(digi.xbee.packets.socket.SocketOptionResponsePacket attribute), 316 attribute), 385
frame_id(digi.xbee.packets.digimesh.RouteInformationPacketne_id(digi.xbee.packets.socket.SocketReceiveFromPacket attribute), 320 attribute), 420
frame id(digi.xbee.packets.filesystem.FSRequestPacket frame id(digi.xbee.packets.socket.SocketReceivePacket

attribute), 417	fs_id (digi.xbee.models.filesystem.FileIdCmd attribute), 135
<pre>frame_id (digi.xbee.packets.socket.SocketSendPacket</pre>	fs_id (digi.xbee.models.filesystem.FileIdNameCmd at-
$\verb frame_id  (digi.xbee.packets.socket.SocketSendToPacket $	tribute), 137
<pre>attribute), 405 frame_id (digi.xbee.packets.socket.SocketStatePacket</pre>	fs_id(digi.xbee.models.filesystem.GetPathIdCmdRequest attribute), 171
attribute), 423	fs_id (digi.xbee.models.filesystem.GetPathIdCmdResponse
frame_id(digi.xbee.packets.wifi.IODataSampleRxIndica	
attribute), 427	$\verb fs_id  (digi.x bee.models. file system. Hash File Cmd Request$
frame_id(digi.xbee.packets.wifi.RemoteATCommandRe.	
attribute), 434	fs_id(digi.xbee.models.filesystem.OpenDirCmdRequest
<pre>frame_id(digi.xbee.packets.wifi.RemoteATCommandWij attribute), 431</pre>	fs_id (digi.xbee.models.filesystem.OpenDirCmdResponse
frame_id(digi.xbee.packets.zigbee.CreateSourceRouteP	
attribute), 450	fs_id(digi.xbee.models.filesystem.OpenFileCmdRequest
$\verb frame_id  (\textit{digi.xbee.packets.zigbee.OTAFirmwareUpdate}) $	
attribute), 452	fs_id(digi.xbee.models.filesystem.OpenFileCmdResponse
frame_id(digi.xbee.packets.zigbee.RegisterDeviceStatus attribute), 442	sPacket attribute), 141 fs_id(digi.xbee.models.filesystem.ReadDirCmdRequest
frame_id(digi.xbee.packets.zigbee.RegisterJoiningDevid	
attribute), 439	fs_id(digi.xbee.models.filesystem.ReadDirCmdResponse
$\verb frame_id  (digi.xbee.packets.zigbee.RouteRecordIndicated)                                      $	
attribute), 446	$\verb fs_id  (digi.xbee.models.filesystem.ReadFileCmdRequest $
FrameError (class in digi.xbee.models.status), 221	attribute), 146
	fs_id(digi.xbee.models.filesystem.ReadFileCmdResponse
digi.xbee.packets.devicecloud), 309 freq_band (digi.xbee.models.zdo.NodeDescriptor at-	attribute), 148
tribute), 228	attribute), 174
	dræss_id (digi.xbee.models.filesystem.WriteFileCmdRequest
class method), 196	attribute), 150
	dr <u>ess_id (digi.xbee.models.filesystem.WriteFileCmdResponse</u>
class method), 197	attribute), 152
from_data() (digi.xbee.filesystem.FileSystemElement	
<pre>static method), 938 from_hex_string()</pre>	FSCmdType (class in digi.xbee.models.filesystem), 131 FSCommandStatus (class in digi.xbee.models.status),
(digi.xbee.models.address.XBee16BitAddress	226
class method), 196	FSRequestPacket (class in
<pre>from_hex_string()</pre>	digi.xbee.packets.filesystem), 323
(digi.xbee.models.address.XBee64BitAddress	FSResponsePacket (class in
class method), 197	digi.xbee.packets.filesystem), 326
from_string() (digi.xbee.models.address.XBeeIMEIA	
<pre>class method), 198 fs_entries (digi.xbee.models.filesystem.OpenDirCmdR</pre>	full_path(digi.xbee.models.filesystem.GetPathIdCmdResponse
attribute), 161	esponse unione), 172
fs_entries(digi.xbee.models.filesystem.ReadDirCmdR	e <b>G</b> onse
attribute), 168	GenericXBeePacket (class in
$\verb fs_id  (\textit{digi.xbee.models.filesystem.CloseDirCmdRequest} $	digi.xbee.packets.base), 242
attribute), 164	get () (digi.xbee.reader.XBeeQueue method), 1001
fs_id(digi.xbee.models.filesystem.CloseFileCmdRequest	300_10010_aaa1 () (a3
attribute), 143 fs. id (digixbee models filesystem CreateDirCmdReaues)	<pre>method), 472 t get_16bit_addr() (digi.xbee.devices.CellularDevice</pre>
attribute), 157	get_16b1t_addr() (aigi.xbee.aevices.CeiluiarDevice method), 678
$\verb fs_id  (\textit{digi.xbee.models.filesystem.DeleteCmdRequest} $	get_16bit_addr() (digi.xbee.devices.DigiMeshDevice
attribute), 178	method), 553

```
get_16bit_addr() (digi.xbee.devices.DigiPointDevice get_64bit_addr()
                                                                          (digi.xbee.devices.XBeeDevice
        method), 584
                                                            method), 498
get_16bit_addr()
                         (digi.xbee.devices.IPDevice
                                                   get 64bit addr() (digi.xbee.devices.ZigBeeDevice
        method), 646
                                                            method), 622
get_16bit_addr() (digi.xbee.devices.LPWANDevice
                                                   get_access_point()
                                                            (digi.xbee.devices.WiFiDevice method), 749
        method), 703
                     (digi.xbee.devices.NBIoTDevice
                                                   get access point timeout()
get_16bit_addr()
                                                            (digi.xbee.devices.WiFiDevice method), 752
        method), 728
get_16bit_addr() (digi.xbee.devices.Raw802Device get_adc_value() (digi.xbee.devices.AbstractXBeeDevice
        method), 521
                                                            method), 477
get_16bit_addr() (digi.xbee.devices.RemoteDigiMeshDeviceadc_value() (digi.xbee.devices.CellularDevice
        method), 820
                                                            method), 679
get_16bit_addr() (digi.xbee.devices.RemoteDigiPointDeviceadc_value() (digi.xbee.devices.DigiMeshDevice
        method), 838
                                                            method), 553
get_16bit_addr() (digi.xbee.devices.RemoteRaw802Dgeice_adc_value() (digi.xbee.devices.DigiPointDevice
        method), 801
                                                            method), 585
get_16bit_addr() (digi.xbee.devices.RemoteXBeeDevicet_adc_value()
                                                                             (digi.xbee.devices.IPDevice
        method), 785
                                                            method), 653
get_16bit_addr() (digi.xbee.devices.RemoteZigBeeDevice_adc_value()
                                                                        (digi.xbee.devices.LPWANDevice
        method), 858
                                                            method), 703
get_16bit_addr()
                       (digi.xbee.devices.WiFiDevice
                                                   get_adc_value()
                                                                         (digi.xbee.devices.NBIoTDevice
        method), 760
                                                            method), 729
get_16bit_addr()
                      (digi.xbee.devices.XBeeDevice
                                                   get adc value() (digi.xbee.devices.Raw802Device
        method), 498
                                                            method), 521
get_16bit_addr() (digi.xbee.devices.ZigBeeDevice
                                                   get_adc_value() (digi.xbee.devices.RemoteDigiMeshDevice
        method), 622
                                                            method), 820
get_64bit_addr() (digi.xbee.devices.AbstractXBeeDevicet_adc_value() (digi.xbee.devices.RemoteDigiPointDevice
        method), 472
                                                            method), 838
get_64bit_addr() (digi.xbee.devices.CellularDevice get_adc_value() (digi.xbee.devices.RemoteRaw802Device
        method), 672
                                                            method), 802
get_64bit_addr() (digi.xbee.devices.DigiMeshDeviceget_adc_value() (digi.xbee.devices.RemoteXBeeDevice
        method), 553
                                                            method), 785
get_64bit_addr() (digi.xbee.devices.DigiPointDevice get_adc_value() (digi.xbee.devices.RemoteZigBeeDevice
        method), 585
                                                            method), 858
get_64bit_addr()
                         (digi.xbee.devices.IPDevice get_adc_value()
                                                                           (digi.xbee.devices.WiFiDevice
        method), 653
                                                            method), 761
get_64bit_addr() (digi.xbee.devices.LPWANDevice
                                                   get adc value()
                                                                           (digi.xbee.devices.XBeeDevice
        method), 703
                                                            method), 498
get_64bit_addr() (digi.xbee.devices.NBIoTDevice
                                                   get adc value()
                                                                         (digi.xbee.devices.ZigBeeDevice
        method), 729
                                                            method), 622
get_64bit_addr() (digi.xbee.devices.Raw802Device get_ai_status() (digi.xbee.devices.Raw802Device
        method), 521
                                                            method), 512
get_64bit_addr() (digi.xbee.devices.RemoteDigiMeshthericai_status() (digi.xbee.devices.RemoteRaw802Device
        method), 820
                                                            method), 799
get_64bit_addr() (digi.xbee.devices.RemoteDigiPointDeviceai_status() (digi.xbee.devices.RemoteZigBeeDevice
        method), 838
                                                            method), 854
get_64bit_addr() (digi.xbee.devices.RemoteRaw802Dqeice_ai_status()
                                                                         (digi.xbee.devices.ZigBeeDevice
        method), 801
                                                            method), 606
get_64bit_addr() (digi.xbee.devices.RemoteXBeeDeviget_analog_value()
                                                                                 (digi.xbee.io.IOSample
        method), 785
                                                            method), 962
get_64bit_addr() (digi.xbee.devices.RemoteZigBeeDegice_api_output_mode()
                                                            (digi.xbee.devices.AbstractXBeeDevice
        method), 858
get_64bit_addr()
                       (digi.xbee.devices.WiFiDevice
                                                            method), 480
        method), 761
                                                   get_api_output_mode()
```

	(digi.xbee.devices.CellularDevice metho	od), ge				_value() <i>PDevice method</i>	<i>I</i> ), 654
get_a	<pre>pi_output_mode()   (digi.xbee.devices.DigiMeshDevice metho</pre>		(			value() .PWANDevice	method),
get_a	<pre>pi_output_mode()   (digi.xbee.devices.DigiPointDevice metho</pre>	_	_ (	_		_value() <i> BIoTDevice</i>	method),
	pi_output_mode() (digi.xbee.devices.IPDevice method), 654	ge	(			_value() Raw802Device	method),
get_a	<pre>pi_output_mode()   (digi.xbee.devices.LPWANDevice methor 704</pre>	od), ge	et_api	_output		_value() RemoteDigiMesh	nDevice
	<pre>pi_output_mode()    (digi.xbee.devices.NBIoTDevice</pre>	od), ge	et_api_ (		_mode_ . <i>devices.R</i>	_value() RemoteDigiPoin	tDevice
	(digi.xbee.devices.Raw802Device metho 522	od), ge	et_api (	_output	mode_ . <i>devices.R</i>	_value() <i>RemoteRaw802L</i>	Device
	<pre>pi_output_mode()   (digi.xbee.devices.RemoteDigiMeshDevice   method), 821 pi_output_mode()</pre>	ge	et_api (	_output	mode_ . <i>devices.R</i>	_value() RemoteXBeeDev	ice
y <u>-</u>	(digi.xbee.devices.RemoteDigiPointDevice method), 839	ge	et_api_	_output	_mode_	_value() RemoteZigBeeDo	evice
get_a	<pre>pi_output_mode()   (digi.xbee.devices.RemoteRaw802Device   method), 802</pre>	ge	et_api_		_mode_	_value() <i>ViFiDevice meth</i>	od), 762
get_a	<pre>pi_output_mode()   (digi.xbee.devices.RemoteXBeeDevice   method), 786</pre>	ge	et_api_ (	_output	_mode_	_value() {BeeDevice	method),
get_a	<pre>pi_output_mode()   (digi.xbee.devices.RemoteZigBeeDevice   method), 859</pre>	ge	(			value() <i>ligBeeDevice</i>	method),
	pi_output_mode()     (digi.xbee.devices.WiFiDevice method), 761		(	digi.xbee.	.reader.Pa	received_ca ucketListener me	
get_a	<pre>pi_output_mode()   (digi.xbee.devices.XBeeDevice</pre>		(			ldr () AbstractXBeeDe	vice
get_a	<pre>pi_output_mode()   (digi.xbee.devices.ZigBeeDevice metho 623</pre>		(		_mac_ad .devices.C	ldr () <i>CellularDevice</i>	method),
get_a	<pre>pi_output_mode_value()   (digi.xbee.devices.AbstractXBeeDevice   method), 480</pre>	ge	(		_mac_ad .devices.L	ldr() <i>DigiMeshDevice</i>	method),
get_a	pi_output_mode_value() (digi.xbee.devices.CellularDevice metho 679		(		_mac_ad .devices.D	ldr () <i>DigiPointDevice</i>	method),
get_a	<pre>pi_output_mode_value()   (digi.xbee.devices.DigiMeshDevice metho 554</pre>	(d),	(	digi.xbee.	_mac_ad . <i>devices.II</i> _mac_ad	PDevice method	<i>I</i> ), 654
get_a	<pre>pi_output_mode_value()   (digi.xbee.devices.DigiPointDevice method</pre>	(d),	7	digi.xbee. 705		.PWANDevice	method),

```
(digi.xbee.devices.NBIoTDevice
                                                                  method),
                                                                                               method), 252
             730
                                                                                  get_checksum() (digi.xbee.packets.common.ATCommPacket
get_bluetooth_mac_addr()
                                                                                               method), 255
             (digi.xbee.devices.Raw802Device
                                                                                  get_checksum() (digi.xbee.packets.common.ATCommQueuePacket
                                                                  method),
                                                                                                method), 258
                                                                                  get checksum() (digi.xbee.packets.common.ATCommResponsePacket
get_bluetooth_mac_addr()
             (digi.xbee.devices.RemoteDigiMeshDevice
                                                                                                method), 262
             method), 822
                                                                                  get_checksum() (digi.xbee.packets.common.ExplicitAddressingPacket
get_bluetooth_mac_addr()
                                                                                                method), 292
             (digi.xbee.devices.RemoteDigiPointDevice
                                                                                  get_checksum() (digi.xbee.packets.common.ExplicitRXIndicatorPacket
             method), 840
                                                                                               method), 296
get_bluetooth_mac_addr()
                                                                                  get_checksum() (digi.xbee.packets.common.IODataSampleRxIndicator
             (digi.xbee.devices.RemoteRaw802Device
                                                                                               method), 289
                                                                                  get_checksum() (digi.xbee.packets.common.ModemStatusPacket
             method), 803
get_bluetooth_mac_addr()
                                                                                                method), 284
             (digi.xbee.devices.RemoteXBeeDevice
                                                                                  get_checksum() (digi.xbee.packets.common.ReceivePacket
             method), 786
                                                                                                method), 265
get_bluetooth_mac_addr()
                                                                                  get_checksum() (digi.xbee.packets.common.RemoteATCommandPacket
             (digi.xbee.devices.RemoteZigBeeDevice
                                                                                                method), 269
             method), 860
                                                                                  get_checksum() (digi.xbee.packets.common.RemoteATCommandResponder)
get_bluetooth_mac_addr()
                                                                                                method), 273
             (digi.xbee.devices.WiFiDevice method), 762
                                                                                  get_checksum() (digi.xbee.packets.common.TransmitPacket
get_bluetooth_mac_addr()
                                                                                                method), 278
                                                                                  \verb"get_checksum"()" \textit{ (digi.xbee.packets.common.TransmitStatusPacket)}
             (digi.xbee.devices.XBeeDevice
                                                                  method).
             500
                                                                                               method), 281
                                                                                  \verb"get_checksum"() \textit{ (digi.xbee.packets.devicecloud.DeviceRequestPacket)}
get_bluetooth_mac_addr()
             (digi.xbee.devices.ZigBeeDevice
                                                                  method),
                                                                                                method), 301
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.DeviceResponsePacket
get_by_id() (digi.xbee.reader.XBeeQueue method),
                                                                                               method), 304
             1002
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.DeviceResponseStatus.
get_by_ip() (digi.xbee.reader.XBeeQueue method),
                                                                                                method), 307
             1002
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.FrameErrorPacket
get_by_remote()
                                      (digi.xbee.reader.XBeeQueue
                                                                                                method), 310
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestSendDataRequestPackets.devicecloud.SendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendSendDataRequestSendDataRequestSendDataRequestS
             method), 1001
get_cellular_ai_status()
                                                                                                method), 314
             (digi.xbee.devices.CellularDevice
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.SendDataResponsePac
                                                                  method),
             670
                                                                                                method), 316
get_cellular_ai_status()
                                                                                  get_checksum() (digi.xbee.packets.digimesh.RouteInformationPacket
             (digi.xbee.devices.LPWANDevice
                                                                                                method), 320
                                                                  method),
             705
                                                                                  get_checksum() (digi.xbee.packets.filesystem.FSRequestPacket
get_cellular_ai_status()
                                                                                               method), 324
             (digi.xbee.devices.NBIoTDevice
                                                                  method),
                                                                                  get_checksum() (digi.xbee.packets.filesystem.FSResponsePacket
                                                                                                method), 327
get_checksum() (digi.xbee.packets.base.GenericXBeePacket_checksum() (digi.xbee.packets.filesystem.RemoteFSRequestPacket
             method), 243
                                                                                                method), 331
get_checksum() (digi.xbee.packets.base.UnknownXBeeBacketchecksum() (digi.xbee.packets.filesystem.RemoteFSResponsePacket
             method), 246
                                                                                                method), 333
get_checksum() (digi.xbee.packets.base.XBeeAPIPacketget_checksum() (digi.xbee.packets.network.RXIPv4Packet
             method), 241
                                                                                                method), 338
get_checksum() (digi.xbee.packets.base.XBeePacket get_checksum() (digi.xbee.packets.network.TXIPv4Packet
             method), 238
                                                                                                method), 341
get checksum() (digi.xbee.packets.cellular.RXSMSPacket checksum() (digi.xbee.packets.raw.RX16IOPacket
             method), 249
                                                                                                method), 364
get_checksum() (digi.xbee.packets.cellular.TXSMSPackget_checksum() (digi.xbee.packets.raw.RX16Packet
```

```
method), 358
                                                             method), 450
get_checksum() (digi.xbee.packets.raw.RX64IOPacket get_checksum() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusI
        method), 362
                                                             method), 452
get_checksum() (digi.xbee.packets.raw.RX64Packet get_checksum() (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
        method), 355
                                                             method), 442
get checksum() (digi.xbee.packets.raw.TX16Packet
                                                    get checksum() (digi.xbee.packets.zigbee.RegisterJoiningDevicePacke
        method), 348
                                                             method), 439
get_checksum() (digi.xbee.packets.raw.TX64Packet get_checksum() (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
        method), 345
                                                             method), 446
get_checksum() (digi.xbee.packets.raw.TXStatusPacketget_comm_iface() (digi.xbee.devices.RemoteDigiMeshDevice
        method), 351
                                                             method), 822
get_checksum() (digi.xbee.packets.relay.UserDataRelay@utputRanketface() (digi.xbee.devices.RemoteDigiPointDevice
        method), 372
                                                             method), 840
get_checksum() (digi.xbee.packets.relay.UserDataRelayRacketomm_iface() (digi.xbee.devices.RemoteRaw802Device
        method), 369
                                                              method), 803
get_checksum() (digi.xbee.packets.socket.SocketBindLisyenPacketm_iface() (digi.xbee.devices.RemoteXBeeDevice
        method), 408
                                                             method), 782
qet_checksum() (digi.xbee.packets.socket.SocketClosePacket_comm_iface() (digi.xbee.devices.RemoteZigBeeDevice
        method), 395
                                                             method), 860
get checksum() (digi.xbee.packets.socket.SocketCloseResponseRawkettions()
        method), 398
                                                              (digi.xbee.devices.DigiMeshNetwork method),
get_checksum() (digi.xbee.packets.socket.SocketConnectPacket 912
        method), 389
                                                     get connections()
get_checksum() (digi.xbee.packets.socket.SocketConnectRespons@Aggiketee.devices.DigiPointNetwork method),
                                                              924
        method), 392
get_checksum() (digi.xbee.packets.socket.SocketCreateBacketconnections()
                                                             (digi.xbee.devices.Raw802Network
                                                                                                method),
        method), 375
get_checksum() (digi.xbee.packets.socket.SocketCreateResponsePacket
        method), 378
                                                     get_connections()
get_checksum() (digi.xbee.packets.socket.SocketListenResponsePatigatxbee.devices.XBeeNetwork
                                                                                                method),
        method), 411
                                                              884
get_checksum() (digi.xbee.packets.socket.SocketNewIPv4ClientPackettions()
        method), 414
                                                             (digi.xbee.devices.ZigBeeNetwork
                                                                                                method),
get_checksum() (digi.xbee.packets.socket.SocketOptionRequestPacket
                                                     get current directory()
        method), 382
get_checksum() (digi.xbee.packets.socket.SocketOptionResponsePdigiexbee.filesystem.LocalXBeeFileSystemManager
        method), 385
                                                             method), 951
get_checksum() (digi.xbee.packets.socket.SocketReceiveFromPacketent_frame_id()
        method), 420
                                                              (digi.xbee.devices.AbstractXBeeDevice
get_checksum() (digi.xbee.packets.socket.SocketReceivePacket method), 472
        method), 417
                                                     get current frame id()
get_checksum() (digi.xbee.packets.socket.SocketSendPacket
                                                              (digi.xbee.devices.CellularDevice
                                                                                                method),
        method), 401
get_checksum() (digi.xbee.packets.socket.SocketSendToPacketurrent_frame_id()
        method), 405
                                                             (digi.xbee.devices.DigiMeshDevice
                                                                                                method),
get_checksum() (digi.xbee.packets.socket.SocketStatePacket
                                                              555
        method), 423
                                                     get_current_frame_id()
get_checksum() (digi.xbee.packets.wifi.IODataSampleRxIndicatonWifiRhekalevices.DigiPointDevice
                                                                                                method),
        method), 428
                                                              587
get_checksum() (digi.xbee.packets.wifi.RemoteATCommantdResponserWifiPacket_id()
        method), 434
                                                              (digi.xbee.devices.IPDevice method), 655
get_checksum() (digi.xbee.packets.wifi.RemoteATCommyotdWifiPackett_frame_id()
        method), 431
                                                              (digi.xbee.devices.LPWANDevice
                                                                                                method).
get checksum() (digi.xbee.packets.zigbee.CreateSourceRoutePack@5
```

```
925
get_current_frame_id()
        (digi.xbee.devices.NBIoTDevice
                                                  get_deep_discovery_timeouts()
                                        method).
                                                          (digi.xbee.devices.Raw802Network
                                                                                           method),
get_current_frame_id()
        (digi.xbee.devices.Raw802Device
                                        method),
                                                  get_deep_discovery_timeouts()
                                                           (digi.xbee.devices.XBeeNetwork
                                                                                           method),
get_current_frame_id()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                  get_deep_discovery_timeouts()
        method), 822
                                                           (digi.xbee.devices.ZigBeeNetwork
                                                                                           method),
                                                           890
get_current_frame_id()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                  get_dest_address()
        method), 840
                                                           (digi.xbee.devices.AbstractXBeeDevice
get_current_frame_id()
                                                          method), 473
        (digi.xbee.devices.RemoteRaw802Device
                                                  get_dest_address()
        method), 804
                                                          (digi.xbee.devices.CellularDevice
                                                                                           method),
get_current_frame_id()
                                                           680
        (digi.xbee.devices.RemoteXBeeDevice
                                                  get_dest_address()
        method), 787
                                                          (digi.xbee.devices.DigiMeshDevice
                                                                                           method),
get_current_frame_id()
                                                           555
                                                  get_dest_address()
        (digi.xbee.devices.RemoteZigBeeDevice
        method), 860
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                           method),
get_current_frame_id()
        (digi.xbee.devices.WiFiDevice method), 762
                                                  get_dest_address()
                                                                           (digi.xbee.devices.IPDevice
get_current_frame_id()
                                                          method), 646
        (digi.xbee.devices.XBeeDevice
                                        method),
                                                  get_dest_address()
                                                          (digi.xbee.devices.LPWANDevice
                                                                                           method),
get_current_frame_id()
        (digi.xbee.devices.ZigBeeDevice
                                        method),
                                                  get_dest_address()
                                                          (digi.xbee.devices.NBIoTDevice
                                                                                           method),
get_data_queue() (digi.xbee.reader.PacketListener
                                                           731
        method), 993
                                                  get_dest_address()
get_data_received_callbacks()
                                                           (digi.xbee.devices.Raw802Device
                                                                                           method),
        (digi.xbee.reader.PacketListener
                                                           523
                                        method),
                                                  get_dest_address()
get_deep_discovery_options()
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
        (digi.xbee.devices.DigiMeshNetwork method),
                                                          method), 822
        913
                                                  get_dest_address()
get_deep_discovery_options()
                                                          (digi.xbee.devices.RemoteDigiPointDevice
        (digi.xbee.devices.DigiPointNetwork method),
                                                          method), 840
        924
                                                  get_dest_address()
get_deep_discovery_options()
                                                          (digi.xbee.devices.RemoteRaw802Device
        (digi.xbee.devices.Raw802Network
                                                          method), 804
                                        method),
                                                  get_dest_address()
                                                          (digi.xbee.devices.RemoteXBeeDevice
get_deep_discovery_options()
        (digi.xbee.devices.XBeeNetwork
                                                          method), 787
                                        method),
        880
                                                  get_dest_address()
get_deep_discovery_options()
                                                          (digi.xbee.devices.RemoteZigBeeDevice
        (digi.xbee.devices.ZigBeeNetwork
                                                          method), 860
                                        method),
                                                  get_dest_address()
                                                           (digi.xbee.devices.WiFiDevice method), 763
get_deep_discovery_timeouts()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                  get_dest_address()
                                                          (digi.xbee.devices.XBeeDevice
        913
                                                                                           method),
get_deep_discovery_timeouts()
                                                           500
        (digi.xbee.devices.DigiPointNetwork method), get dest address()
```

(digi.xbee.devices.ZigBeeDevice 624 get_dest_ip_addr()	method),	<pre>get_device_by_node_id()      (digi.xbee.devices.XBeeNetwork method),      883</pre>
(digi.xbee.devices.CellularDevice 680	method),	<pre>get_device_by_node_id()      (digi.xbee.devices.ZigBeeNetwork method),</pre>
<pre>get_dest_ip_addr() (digi.xbee.devices.</pre>	.IPDevice	891 get_devices() (digi.xbee.devices.DigiMeshNetwork method), 914
(digi.xbee.devices.LPWANDevice 705	method),	get_devices() (digi.xbee.devices.DigiPointNetwork method), 926
get_dest_ip_addr() (digi.xbee.devices.NBIoTDevice 731	method),	<pre>get_devices() (digi.xbee.devices.Raw802Network</pre>
get_dest_ip_addr() (digi.xbee.devices.WiFiDevice method	d), 763	method), 876 get_devices() (digi.xbee.devices.ZigBeeNetwork
<pre>get_device_by_16()      (digi.xbee.devices.DigiMeshNetwork 914</pre>	method),	<pre>method), 891 get_digital_value() (digi.xbee.io.IOSample     method), 961</pre>
<pre>get_device_by_16()      (digi.xbee.devices.DigiPointNetwork)</pre>	method),	get_dio_value() (digi.xbee.devices.AbstractXBeeDevice method), 479
925 get_device_by_16() (digi.xbee.devices.Raw802Network	method),	<pre>get_dio_value() (digi.xbee.devices.CellularDevice</pre>
902 get_device_by_16() (digitable devices YPacNatural)	method),	method), 555 get_dio_value() (digi.xbee.devices.DigiPointDevice
(digi.xbee.devices.XBeeNetwork 883 get_device_by_16()	meinoa),	<pre>method), 587 get_dio_value()</pre>
(digi.xbee.devices.ZigBeeNetwork 891	method),	<pre>get_dio_value() (digi.xbee.devices.LPWANDevice</pre>
<pre>get_device_by_64()      (digi.xbee.devices.DigiMeshNetwork 914</pre>	method),	<pre>get_dio_value() (digi.xbee.devices.NBIoTDevice</pre>
<pre>get_device_by_64()      (digi.xbee.devices.DigiPointNetwork</pre>	method),	<pre>method), 523 get_dio_value() (digi.xbee.devices.RemoteDigiMeshDevic     method), 823</pre>
get_device_by_64() (digi.xbee.devices.Raw802Network 903	method),	<pre>get_dio_value() (digi.xbee.devices.RemoteDigiPointDevice</pre>
get_device_by_64() (digi.xbee.devices.XBeeNetwork 883	method),	method), 804 get_dio_value() (digi.xbee.devices.RemoteXBeeDevice method), 787
get_device_by_64() (digi.xbee.devices.ZigBeeNetwork 891	method),	<pre>get_dio_value() (digi.xbee.devices.RemoteZigBeeDevice</pre>
get_device_by_node_id() (digi.xbee.devices.DigiMeshNetwork 914	method),	get_dio_value() (digi.xbee.devices.WiFiDevice method), 763 get_dio_value() (digi.xbee.devices.XBeeDevice method), 501
get_device_by_node_id() (digi.xbee.devices.DigiPointNetwork 925	method),	get_dio_value() (digi.xbee.devices.ZigBeeDevice method), 625 get_discovery_callbacks()
get_device_by_node_id() (digi.xbee.devices.Raw802Network 903	method),	(digi.xbee.devices.DigiMeshNetwork method), 914 get_discovery_callbacks()

```
(digi.xbee.devices.DigiPointNetwork method),
                                                  get_file_hash() (digi.xbee.filesystem.LocalXBeeFileSystemManager
        926
                                                           method), 953
get_discovery_callbacks()
                                                   get file manager()
        (digi.xbee.devices.Raw802Network
                                                           (digi.xbee.devices.AbstractXBeeDevice
                                         method),
                                                           method), 484
get discovery callbacks()
                                                   get file manager()
        (digi.xbee.devices.XBeeNetwork
                                                           (digi.xbee.devices.CellularDevice
                                                                                            method).
                                         method).
                                                           681
get_discovery_callbacks()
                                                  get_file_manager()
        (digi.xbee.devices.ZigBeeNetwork
                                                           (digi.xbee.devices.DigiMeshDevice
                                         method),
                                                                                            method),
get_discovery_options()
                                                  get_file_manager()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                            method),
        915
get_discovery_options()
                                                                            (digi.xbee.devices.IPDevice
                                                  get_file_manager()
        (digi.xbee.devices.DigiPointNetwork method),
                                                           method), 655
        926
                                                  get_file_manager()
get_discovery_options()
                                                           (digi.xbee.devices.LPWANDevice
                                                                                            method),
        (digi.xbee.devices.Raw802Network
                                         method),
                                                  get file manager()
get_discovery_options()
                                                           (digi.xbee.devices.NBIoTDevice
                                                                                            method),
        (digi.xbee.devices.XBeeNetwork
                                         method),
        879
                                                  get_file_manager()
get_discovery_options()
                                                           (digi.xbee.devices.Raw802Device
                                                                                            method).
                                                           524
        (digi.xbee.devices.ZigBeeNetwork
                                         method),
                                                  get_file_manager()
get_discovery_timeout()
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
        (digi.xbee.devices.DigiMeshNetwork method),
                                                           method), 823
                                                   get_file_manager()
                                                           (digi.xbee.devices.RemoteDigiPointDevice
get_discovery_timeout()
        (digi.xbee.devices.DigiPointNetwork method),
                                                           method), 841
        926
                                                  get_file_manager()
                                                           (digi.xbee.devices.RemoteRaw802Device
get_discovery_timeout()
        (digi.xbee.devices.Raw802Network
                                                           method), 805
                                         method),
                                                  get_file_manager()
                                                           (digi.xbee.devices.RemoteXBeeDevice
get_discovery_timeout()
        (digi.xbee.devices.XBeeNetwork
                                         method),
                                                           method), 788
                                                  get_file_manager()
get_discovery_timeout()
                                                           (digi.xbee.devices.RemoteZigBeeDevice
        (digi.xbee.devices.ZigBeeNetwork
                                                           method), 861
                                         method),
                                                  get file manager()
get_dns_address() (digi.xbee.devices.WiFiDevice
                                                           (digi.xbee.devices.WiFiDevice method), 763
        method), 754
                                                  get file manager()
                                                           (digi.xbee.devices.XBeeDevice
get_explicit_data_received_callbacks()
                                                                                            method),
        (digi.xbee.reader.PacketListener method), 999
                                                           501
get_explicit_queue()
                                                  get_file_manager()
        (digi.xbee.reader.PacketListener
                                                           (digi.xbee.devices.ZigBeeDevice
                                         method),
                                                                                            method),
        994
                                                           625
get_file() (digi.xbee.fileSystem.FileSystemManager
                                                  get_file_ymodem()
                                                                                 (in
                                                                                             module
        method), 941
                                                           digi.xbee.util.xmodem), 463
get_file() (digi.xbee.filesystem.LocalXBeeFileSystemMgmtgefirmware_version()
                                                           (digi.xbee.devices.AbstractXBeeDevice
        method), 952
get_file_hash() (digi.xbee.filesystem.FileSystemManager
                                                           method), 471
        method), 943
                                                   get firmware version()
```

(digi.xbee.devices.CellularDevice method), 681	<pre>get_frame_spec_data()      (digi.xbee.packets.cellular.RXSMSPacket</pre>
<pre>get_firmware_version()</pre>	method), 249
(digi.xbee.devices.DigiMeshDevice method), 556	<pre>get_frame_spec_data()      (digi.xbee.packets.cellular.TXSMSPacket</pre>
<pre>get_firmware_version()</pre>	method), 252
(digi.xbee.devices.DigiPointDevice method),	<pre>get_frame_spec_data()</pre>
588	(digi.xbee.packets.common.ATCommPacket
<pre>get_firmware_version()</pre>	method), 255
(digi.xbee.devices.IPDevice method), 655	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.x bee.packets.common. ATCommQueue Packet
(digi.xbee.devices.LPWANDevice method),	method), 258
706	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.xbee.packets.common.ATCommResponsePacket
(digi.xbee.devices.NBIoTDevice method),	method), 262
732	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.xbee.packets.common.ExplicitAddressingPacket
(digi.xbee.devices.Raw802Device method),	method), 292
524	get_frame_spec_data()
get_firmware_version()	(digi.xbee.packets.common.ExplicitRXIndicatorPacket
(digi.xbee.devices.RemoteDigiMeshDevice	method), 296
<pre>method), 823 get_firmware_version()</pre>	get_frame_spec_data()
(digi.xbee.devices.RemoteDigiPointDevice	(digi.xbee.packets.common.IODataSampleRxIndicatorPacket method), 289
method), 841	get_frame_spec_data()
get_firmware_version()	(digi.xbee.packets.common.ModemStatusPacket
(digi.xbee.devices.RemoteRaw802Device	method), 285
method), 805	get_frame_spec_data()
<pre>get_firmware_version()</pre>	(digi.xbee.packets.common.ReceivePacket
(digi.xbee.devices.RemoteXBeeDevice	method), 266
method), 788	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.xbee.packets.common.RemoteATCommandPacket
(digi.xbee.devices.RemoteZigBeeDevice	method), 270
method), 862	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.x bee.packets.common. Remote ATC ommand Response Packet
(digi.xbee.devices.WiFiDevice method), 764	method), 274
<pre>get_firmware_version()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.devices.XBeeDevice method),	(digi.xbee.packets.common.TransmitPacket
501	method), 278
<pre>get_firmware_version()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.devices.ZigBeeDevice method),	(digi.xbee.packets.common.TransmitStatusPacket
625	method), 282
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.base.GenericXBeePacket method), 243	(digi.xbee.packets.devicecloud.DeviceRequestPacket method), 301
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.base.UnknownXBeePacket	(digi.xbee.packets.devicecloud.DeviceResponsePacket
method), 246	method), 304
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.base.XBeeAPIPacket	(digi.xbee.packets.devicecloud.DeviceResponseStatusPacket
method), 240	method), 307
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.base.XBeePacket method),	(digi.xbee.packets.devicecloud.FrameErrorPacket
239	method), 310

```
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.devicecloud.SendDataRequestPacket
                                                            (digi.xbee.packets.socket.SocketBindListenPacket
        method), 314
                                                            method), 408
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.devicecloud.SendDataResponsePacket
                                                            (digi.xbee.packets.socket.SocketClosePacket
        method), 316
                                                            method), 395
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.digimesh.RouteInformationPacket
                                                            (digi.xbee.packets.socket.SocketCloseResponsePacket
        method), 321
                                                            method), 399
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.filesystem.FSRequestPacket
                                                            (digi.xbee.packets.socket.SocketConnectPacket
        method), 325
                                                            method), 389
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.filesystem.FSResponsePacket
                                                            (digi.xbee.packets.socket.SocketConnectResponsePacket
        method), 327
                                                             method), 392
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.filesystem.RemoteFSRequestPacket
                                                            (digi.xbee.packets.socket.SocketCreatePacket
        method), 331
                                                            method), 375
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.filesystem.RemoteFSResponsePacket
                                                            (digi.xbee.packets.socket.SocketCreateResponsePacket
        method), 333
                                                            method), 379
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.network.RXIPv4Packet
                                                            (digi.xbee.packets.socket.SocketListenResponsePacket
        method), 338
                                                            method), 411
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.network.TXIPv4Packet
                                                            (digi.xbee.packets.socket.SocketNewIPv4ClientPacket
        method), 341
                                                            method), 415
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX16IOPacket method),
                                                            (digi.xbee.packets.socket.SocketOptionRequestPacket
        364
                                                            method), 382
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX16Packet
                                          method),
                                                             (digi.xbee.packets.socket.SocketOptionResponsePacket
        359
                                                            method), 385
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX64IOPacket method),
                                                             (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                            method), 420
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX64Packet
                                          method),
                                                            (digi.xbee.packets.socket.SocketReceivePacket
        355
                                                            method), 417
                                                    get_frame_spec_data()
get_frame_spec_data()
        (digi.xbee.packets.raw.TX16Packet
                                          method),
                                                            (digi.xbee.packets.socket.SocketSendPacket
        348
                                                            method), 402
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.TX64Packet
                                                            (digi.xbee.packets.socket.SocketSendToPacket
                                          method),
                                                            method), 405
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.TXStatusPacket
                                                            (digi.xbee.packets.socket.SocketStatePacket
        method), 351
                                                            method), 423
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.relay.UserDataRelayOutputPacket
                                                            (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacket
        method), 372
                                                             method), 428
get_frame_spec_data()
                                                    get frame spec data()
                                                            (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket
        (digi.xbee.packets.relay.UserDataRelayPacket
        method), 369
                                                            method), 434
```

(digi.xbee.packets.wifi.RemoteATCommandWifiPacket

get\_frame\_spec\_data()

*method*), 302

```
get_frame_type() (digi.xbee.packets.devicecloud.DeviceResponseState
             method), 432
get_frame_spec_data()
                                                                                                method), 308
             (digi.xbee.packets.zigbee.CreateSourceRoutePacketet_frame_type() (digi.xbee.packets.devicecloud.FrameErrorPacket
             method), 450
                                                                                                method), 311
get_frame_spec_data()
                                                                                  get_frame_type() (digi.xbee.packets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.devicecloud.SendDataRequestPackets.deviceclou
             (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacketmethod), 314
             method), 452
                                                                                  get_frame_type() (digi.xbee.packets.devicecloud.SendDataResponse)
get_frame_spec_data()
                                                                                                method), 316
              (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket \_frame\_type() (digi.xbee.packets.digimesh.RouteInformationPackets)
             method), 442
                                                                                                method), 321
get_frame_spec_data()
                                                                                  get_frame_type() (digi.xbee.packets.filesystem.FSRequestPacket
             (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket
                                                                                                method), 325
             method), 439
                                                                                  get_frame_type() (digi.xbee.packets.filesystem.FSResponsePacket
get_frame_spec_data()
                                                                                                method), 328
             (digi.xbee.packets.zigbee.RouteRecordIndicatorPagket_frame_type() (digi.xbee.packets.filesystem.RemoteFSRequestPac
             method), 446
                                                                                                method), 331
get_frame_type() (digi.xbee.packets.base.GenericXBeq@ackets.ame_type() (digi.xbee.packets.filesystem.RemoteFSResponsePa
             method), 244
                                                                                                method), 333
get_frame_type() (digi.xbee.packets.base.UnknownXBpePacketme_type() (digi.xbee.packets.network.RXIPv4Packet
             method), 246
                                                                                                method), 338
get_frame_type() (digi.xbee.packets.base.XBeeAPIPacket_frame_type() (digi.xbee.packets.network.TXIPv4Packet
             method), 240
                                                                                                method), 341
get_frame_type() (digi.xbee.packets.cellular.RXSMSPacket_frame_type() (digi.xbee.packets.raw.RX16IOPacket
             method), 249
                                                                                                method), 364
get_frame_type() (digi.xbee.packets.cellular.TXSMSPqcket_frame_type() (digi.xbee.packets.raw.RX16Packet
             method), 252
                                                                                                method), 359
get_frame_type() (digi.xbee.packets.common.ATComm@acketrame_type() (digi.xbee.packets.raw.RX64IOPacket
                                                                                                method), 362
             method), 255
get_frame_type() (digi.xbee.packets.common.ATCommgQue_terPawket_type() (digi.xbee.packets.raw.RX64Packet
             method), 258
                                                                                                method), 355
get_frame_type() (digi.xbee.packets.common.ATCommgResppnsaRacketpe() (digi.xbee.packets.raw.TX16Packet
                                                                                                method), 349
             method), 262
get_frame_type() (digi.xbee.packets.common.ExplicitAddressingRacketpe() (digi.xbee.packets.raw.TX64Packet
             method), 292
                                                                                                method), 345
get_frame_type() (digi.xbee.packets.common.ExplicitRXIndicatonPackete() (digi.xbee.packets.raw.TXStatusPacket
                                                                                                method), 352
             method), 296
get_frame_type() (digi.xbee.packets.common.IODataSpanpl&RxIndicatopPackets.relay.UserDataRelayOutputPack
                                                                                                method), 372
             method), 289
get_frame_type() (digi.xbee.packets.common.ModemStatus_Rackete_type() (digi.xbee.packets.relay.UserDataRelayPacket
                                                                                                method), 369
             method), 285
get_frame_type() (digi.xbee.packets.common.ReceiveBacketframe_type() (digi.xbee.packets.socket.SocketBindListenPacket
             method), 266
                                                                                                method), 408
get_frame_type() (digi.xbee.packets.common.RemoteAfEConfirmandPackets() (digi.xbee.packets.socket.SocketClosePacket
             method), 270
                                                                                                method), 396
get_frame_type() (digi.xbee.packets.common.RemoteAfEContinuandResponseRadaketi.xbee.packets.socket.SocketCloseResponsePaci
             method), 274
                                                                                                method), 399
get_frame_type() (digi.xbee.packets.common.Transmit@Atck_drame_type() (digi.xbee.packets.socket.SocketConnectPacket
             method), 278
                                                                                                method), 389
get_frame_type() (digi.xbee.packets.common.TransmiqStatusFranket_type() (digi.xbee.packets.socket.SocketConnectResponseP
             method), 282
                                                                                                method), 393
get_frame_type() (digi.xbee.packets.devicecloud.DevigeReqfirstRacketype() (digi.xbee.packets.socket.SocketCreatePacket
```

get\_frame\_type() (digi.xbee.packets.devicecloud.DeviceResponsePac

method), 305

1056 Index

method), 375

```
get_frame_type() (digi.xbee.packets.socket.SocketCreateResponsePacketse=value()
                  method), 379
                                                                                                                                      (digi.xbee.packets.common.ATCommQueuePacket
get_frame_type() (digi.xbee.packets.socket.istenResponsePthack), 259
                   method), 412
                                                                                                                   get_frame_type_value()
\verb|get_frame_type()| (\textit{digi.xbee.packets.socket.SocketNewIPv4Clief(tRgckbtee.packets.common.ATCommResponsePackets)| (\textit{digi.xbee.packets.socket.packets.common.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atcommon.atco
                  method), 415
                                                                                                                                      method), 262
get_frame_type()(digi.xbee.packets.socket.SocketOptiontRecfuextRacketpe_value()
                   method), 382
                                                                                                                                       (digi.xbee.packets.common.ExplicitAddressingPacket
get_frame_type() (digi.xbee.packets.socket.SocketOptionResponseHwallet293
                  method), 386
                                                                                                                   get_frame_type_value()
\verb|get_frame_type()| (\textit{digi.xbee.packets.socket.SocketReceiveFromRdigixbee.packets.common.ExplicitRXIndicatorPacket)| (\textit{digi.xbee.packets.socket.packets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndicatorPackets.common.explicitRXIndic
                   method), 420
                                                                                                                                      method), 297
qet_frame_type() (digi.xbee.packets.socket.SocketReceivte_dicketme_type_value()
                                                                                                                                       (digi.xbee.packets.common.IODataSampleRxIndicatorPacket
                  method), 417
get_frame_type() (digi.xbee.packets.socket.SocketSendPacket method), 289
                   method), 402
                                                                                                                   get_frame_type_value()
qet_frame_type() (digi.xbee.packets.socket.SocketSendToPacket(digi.xbee.packets.common.ModemStatusPacket
                  method), 405
                                                                                                                                      method), 285
get_frame_type() (digi.xbee.packets.socketStatePacketsrame_type_value()
                   method), 423
                                                                                                                                      (digi.xbee.packets.common.ReceivePacket
get_frame_type() (digi.xbee.packets.wifi.IODataSampleRxIndicatetHWifi)Packet
                  method), 428
                                                                                                                   get_frame_type_value()
get_frame_type() (digi.xbee.packets.wifi.RemoteATCommandResfigirxbWifiRaklets.common.RemoteATCommandPacket
                   method), 434
                                                                                                                                      method), 270
get_frame_type() (digi.xbee.packets.wifi.RemoteATComputa_nfilWifiBacketpe_value()
                  method), 432
                                                                                                                                      (digi.xbee.packets.common.RemoteATCommandResponsePacket
get_frame_type() (digi.xbee.packets.zigbee.CreateSourceRoutePraeRoute), 274
                   method), 450
                                                                                                                   get_frame_type_value()
qet_frame_type() (digi.xbee.packets.zigbee.OTAFirmwareUpda(etsgixthsPackets.common.TransmitPacket
                  method), 453
                                                                                                                                       method), 278
qet_frame_type() (digi.xbee.packets.zigbee.RegisterDeviteSfatusRacketpe_value()
                  method), 442
                                                                                                                                      (digi.xbee.packets.common.TransmitStatusPacket
get_frame_type() (digi.xbee.packets.zigbee.RegisterJoiningDevinePlandlet 282
                  method), 439
                                                                                                                   get_frame_type_value()
get_frame_type() (digi.xbee.packets.zigbee.RouteRecordIndicatchiQiollete.packets.devicecloud.DeviceRequestPacket
                  method), 446
                                                                                                                                      method), 302
get_frame_type_value()
                                                                                                                   get_frame_type_value()
                  (digi.xbee.packets.base.GenericXBeePacket
                                                                                                                                      (digi.xbee.packets.devicecloud.DeviceResponsePacket
                  method), 244
                                                                                                                                      method), 305
get_frame_type_value()
                                                                                                                   get_frame_type_value()
                   (digi.xbee.packets.base.UnknownXBeePacket
                                                                                                                                      (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket
                  method), 246
                                                                                                                                      method), 308
get_frame_type_value()
                                                                                                                   get frame type value()
                  (digi.xbee.packets.base.XBee APIP acket\\
                                                                                                                                      (digi.xbee.packets.devicecloud.FrameErrorPacket
                  method), 240
                                                                                                                                      method), 311
get_frame_type_value()
                                                                                                                   get_frame_type_value()
                   (digi.xbee.packets.cellular.RXSMSPacket
                                                                                                                                      (digi.x bee.packets. device cloud. Send Data Request Packet\\
                  method), 249
                                                                                                                                      method), 314
get_frame_type_value()
                                                                                                                   get_frame_type_value()
                   (digi.xbee.packets.cellular.TXSMSPacket
                                                                                                                                      (digi.xbee.packets.devicecloud.SendDataResponsePacket
                  method), 252
                                                                                                                                      method), 316
get frame type value()
                                                                                                                   get frame type value()
                                                                                                                                      (digi.xbee.packets.digimesh.RouteInformationPacket
                   (digi.xbee.packets.common.ATCommPacket
                  method), 256
                                                                                                                                      method), 321
```

<pre>get_frame_type_value()      (digi.xbee.packets.filesystem.FSRequestPacket      method), 325</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketConnectPacket      method), 390</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.filesystem.FSResponsePacket      method), 328</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketConnectResponsePacket      method), 393</pre>
<pre>get_frame_type_value()     (digi.xbee.packets.filesystem.RemoteFSRequestPackets.filesystem.filesys</pre>	<pre>get_frame_type_value() acket (digi.xbee.packets.socket.SocketCreatePacket</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.filesystem.RemoteFSResponse)      method), 334</pre>	<pre>get_frame_type_value() Packet (digi.xbee.packets.socket.SocketCreateResponsePacket</pre>
<pre>get_frame_type_value()     (digi.xbee.packets.network.RXIPv4Packet     method), 339</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketListenResponsePacket      method), 412</pre>
<pre>get_frame_type_value()     (digi.xbee.packets.network.TXIPv4Packet     method), 342</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketNewIPv4ClientPacket      method), 415</pre>
<pre>get_frame_type_value()     (digi.xbee.packets.raw.RX16IOPacket method),     365</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketOptionRequestPacket      method), 382</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.raw.RX16Packet method),      359</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketOptionResponsePacket      method), 386</pre>
<pre>get_frame_type_value()     (digi.xbee.packets.raw.RX64IOPacket method),     362</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketReceiveFromPacket      method), 420</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.raw.RX64Packet method),      355</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketReceivePacket      method), 417</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.raw.TX16Packet method),      349</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketSendPacket      method), 402</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.raw.TX64Packet method),      346</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketSendToPacket      method), 405</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.raw.TXStatusPacket      method), 352</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketStatePacket      method), 423</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.relay.UserDataRelayOutputPa      method), 372</pre>	<pre>get_frame_type_value()</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.relay.UserDataRelayPacket      method), 370</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket      method), 434</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketBindListenPacket           method), 409</pre>	<pre>get_frame_type_value()</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketClosePacket      method), 396</pre>	<pre>get_frame_type_value()      (digi.xbee.packets.zigbee.CreateSourceRoutePacket      method), 450</pre>
<pre>get_frame_type_value()      (digi.xbee.packets.socket.SocketCloseResponsePacket), 399</pre>	<pre>get_frame_type_value()</pre>

```
get_hardware_version()
get_frame_type_value()
        (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
                                                           (digi.xbee.devices.XBeeDevice
                                                                                            method),
        method), 442
get_frame_type_value()
                                                   get_hardware_version()
        (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket
                                                           (digi.xbee.devices.ZigBeeDevice
                                                                                            method),
        method), 439
                                                   get_hsb() (digi.xbee.models.address.XBee16BitAddress
get_frame_type_value()
        (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
                                                           method), 196
                                                   get_imei_addr() (digi.xbee.devices.CellularDevice
        method), 446
get_fs_frame_received_callbacks()
                                                           method), 671
        (digi.xbee.reader.PacketListener
                                         method),
                                                  get_imei_addr() (digi.xbee.devices.LPWANDevice
                                                           method), 707
                                                                        (digi.xbee.devices.NBIoTDevice
get_gateway_address()
                                                  get_imei_addr()
        (digi.xbee.devices.WiFiDevice method), 754
                                                           method), 732
get_hardware_version()
                                                   get_int_from_byte()
                                                                                   (in
                                                                                             module
        (digi.xbee.devices.AbstractXBeeDevice
                                                           digi.xbee.util.utils), 458
        method), 471
                                                  get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.AbstractXBeeDevice
        (digi.xbee.devices.CellularDevice
                                                           method), 476
                                         method),
                                                  get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.CellularDevice
                                                                                            method),
        (digi.xbee.devices.DigiMeshDevice
                                         method),
        556
                                                   get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.DigiMeshDevice
                                                                                            method),
                                                           556
        (digi.xbee.devices.DigiPointDevice
                                         method),
                                                  get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                            method),
        (digi.xbee.devices.IPDevice method), 656
get_hardware_version()
                                                   get_io_configuration()
                                                           (digi.xbee.devices.IPDevice method), 656
        (digi.xbee.devices.LPWANDevice
                                         method),
        706
                                                   get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.LPWANDevice
                                                                                            method),
        (digi.xbee.devices.NBIoTDevice
                                                           707
                                         method),
                                                  get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.NBIoTDevice
                                                                                            method),
        (digi.xbee.devices.Raw802Device
                                         method).
        524
                                                  get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.Raw802Device
                                                                                            method),
                                                           524
        (digi.xbee.devices.RemoteDigiMeshDevice
        method), 823
                                                   get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
        (digi.xbee.devices.RemoteDigiPointDevice
                                                           method), 824
        method), 841
                                                  get_io_configuration()
                                                           (digi.xbee.devices.RemoteDigiPointDevice
get_hardware_version()
        (digi.xbee.devices.RemoteRaw802Device
                                                           method), 842
        method), 805
                                                  get_io_configuration()
                                                           (digi.xbee.devices.RemoteRaw802Device
get_hardware_version()
        (digi.xbee.devices.RemoteXBeeDevice
                                                           method), 805
        method), 788
                                                  get_io_configuration()
get_hardware_version()
                                                           (digi.xbee.devices.RemoteXBeeDevice
        (digi.xbee.devices.RemoteZigBeeDevice
                                                           method), 788
                                                   get io configuration()
        method), 862
get_hardware_version()
                                                           (digi.xbee.devices.RemoteZigBeeDevice
        (digi.xbee.devices.WiFiDevice method), 764
                                                           method), 862
```

get_io_configuration() (digi.xbee.devices.WiFiDevice method),764	(digi.xbee.devices.ZigBeeDevice method), 626
<pre>get_io_configuration()           (digi.xbee.devices.XBeeDevice</pre>	<pre>get_ip_addr() (digi.xbee.devices.CellularDevice     method), 682</pre>
501 get_io_configuration()	<pre>get_ip_addr() (digi.xbee.devices.IPDevice method), 643</pre>
(digi.xbee.devices.ZigBeeDevice method), 626	get_ip_addr() (digi.xbee.devices.LPWANDevice method), 707
<pre>get_io_sample_received_callbacks()           (digi.xbee.reader.PacketListener method), 999</pre>	<pre>get_ip_addr()</pre>
get_io_sampling_rate() (digi.xbee.devices.AbstractXBeeDevice	get_ip_addr() (digi.xbee.devices.WiFiDevice method), 765
method), 476  get_io_sampling_rate()  (digitable devices Cellular Devices method)	<pre>get_ip_addressing_mode()     (digi.xbee.devices.WiFiDevice method), 752</pre>
(digi.xbee.devices.CellularDevice method), 672 get_io_sampling_rate()	<pre>get_ip_data_received_callbacks()     (digi.xbee.reader.PacketListener method),</pre>
(digi.xbee.devices.DigiMeshDevice method), 557	get_ip_queue() (digi.xbee.reader.PacketListener method), 994
<pre>get_io_sampling_rate()      (digi.xbee.devices.DigiPointDevice method),</pre>	<pre>get_local_file_hash() (in module</pre>
588 get_io_sampling_rate() (digi.xbee.devices.IPDevice method),656	<pre>get_local_xbee_device()     (digi.xbee.devices.RemoteDigiMeshDevice     method), 825</pre>
<pre>get_io_sampling_rate()           (digi.xbee.devices.LPWANDevice method),</pre>	<pre>get_local_xbee_device()      (digi.xbee.devices.RemoteDigiPointDevice      method), 843</pre>
<pre>get_io_sampling_rate()       (digi.xbee.devices.NBIoTDevice method),       733</pre>	get_local_xbee_device() (digi.xbee.devices.RemoteRaw802Device method), 806
<pre>get_io_sampling_rate()      (digi.xbee.devices.Raw802Device method),      525</pre>	<pre>get_local_xbee_device()      (digi.xbee.devices.RemoteXBeeDevice      method), 781</pre>
<pre>get_io_sampling_rate()     (digi.xbee.devices.RemoteDigiMeshDevice     method), 824</pre>	<pre>get_local_xbee_device()      (digi.xbee.devices.RemoteZigBeeDevice      method), 863</pre>
<pre>get_io_sampling_rate()     (digi.xbee.devices.RemoteDigiPointDevice     method), 842</pre>	<pre>get_local_xbee_info()      (digi.xbee.comm_interface.XBeeCommunicationInterface      method), 465</pre>
<pre>get_io_sampling_rate()     (digi.xbee.devices.RemoteRaw802Device     method), 805</pre>	<pre>get_local_xbee_info()      (digi.xbee.serial.XBeeSerialPort method),      1008</pre>
<pre>get_io_sampling_rate()      (digi.xbee.devices.RemoteXBeeDevice</pre>	<pre>get_lsb() (digi.xbee.models.address.XBee16BitAddress     method), 196</pre>
<pre>method), 789 get_io_sampling_rate()</pre>	<pre>get_many_to_one_broadcasting_time()     (digi.xbee.devices.ZigBeeDevice method), 606</pre>
(digi.xbee.devices.RemoteZigBeeDevice method), 862	<pre>get_mask_address()     (digi.xbee.devices.WiFiDevice method), 753</pre>
get_io_sampling_rate() (digi.xbee.devices.WiFiDevice method),764	<pre>get_micropython_data_received_callbacks()      (digi.xbee.reader.PacketListener method), 1000</pre>
<pre>get_io_sampling_rate()       (digi.xbee.devices.XBeeDevice method),       502</pre>	get_modem_status_received_callbacks() (digi.xbee.reader.PacketListener method), 999
get io sampling rate()	<pre>get_neighbor_table()    (digi.xbee.models.zdo.NeighborTableReader</pre>

```
method), 233
                                                            525
get_neighbors() (digi.xbee.devices.DigiMeshDevice get_next_frame_id()
                                                            (digi.xbee.devices.WiFiDevice method), 765
        method), 546
get_neighbors() (digi.xbee.devices.RemoteDigiMeshDevice_next_frame_id()
        method), 817
                                                            (digi.xbee.devices.XBeeDevice
                                                                                              method),
                                                            495
get neighbors() (digi.xbee.devices.RemoteZigBeeDevice
        method), 855
                                                    get_next_frame_id()
get_neighbors() (digi.xbee.devices.ZigBeeDevice
                                                            (digi.xbee.devices.ZigBeeDevice
                                                                                              method),
        method), 615
                                                            626
get_neighbors() (digi.xbee.models.zdo.NeighborFinderget_node_connections()
        method), 235
                                                            (digi.xbee.devices.DigiMeshNetwork method),
get_network()(digi.xbee.comm_interface.XBeeCommunicationIndefface
        method), 465
                                                    get_node_connections()
get_network()
                    (digi.xbee.devices.CellularDevice
                                                            (digi.xbee.devices.DigiPointNetwork method),
                                                            927
        method), 682
get_network()
                   (digi.xbee.devices.DigiMeshDevice
                                                   get_node_connections()
                                                            (digi.xbee.devices.Raw802Network
                                                                                              method),
        method), 557
                   (digi.xbee.devices.DigiPointDevice
                                                            904
get_network()
        method), 589
                                                   get_node_connections()
get_network() (digi.xbee.devices.IPDevice method),
                                                            (digi.xbee.devices.XBeeNetwork
                                                                                              method),
        646
                                                            885
                    (digi.xbee.devices.LPWANDevice
get_network()
                                                   get_node_connections()
                                                            (digi.xbee.devices.ZigBeeNetwork
        method), 707
                                                                                              method),
get_network()
                      (digi.xbee.devices.NBIoTDevice
        method), 733
                                                   get_node_descriptor()
get_network()
                    (digi.xbee.devices.Raw802Device
                                                            (digi.xbee.models.zdo.NodeDescriptorReader
        method), 525
                                                            method), 227
                       (digi.xbee.devices.WiFiDevice
                                                   get_node_id() (digi.xbee.devices.AbstractXBeeDevice
get_network()
        method), 765
                                                            method), 471
                       (digi.xbee.devices.XBeeDevice
                                                                        (digi.xbee.devices.CellularDevice
get_network()
                                                   get_node_id()
        method), 494
                                                            method), 672
get_network()
                     (digi.xbee.devices.ZigBeeDevice
                                                   get_node_id()
                                                                       (digi.xbee.devices.DigiMeshDevice
        method), 626
                                                            method), 558
                     (digi.xbee.serial.XBeeSerialPort
                                                   get_node_id()
                                                                       (digi.xbee.devices.DigiPointDevice
get_network()
        method), 1008
                                                            method), 589
                                                    get_node_id() (digi.xbee.devices.IPDevice method),
get_next_frame_id()
        (digi.xbee.devices.CellularDevice
                                          method),
                                                            657
                                                                         (digi.xbee.devices.LPWANDevice
                                                    get_node_id()
                                                            method), 708
get_next_frame_id()
        (digi.xbee.devices.DigiMeshDevice
                                          method),
                                                                          (digi.xbee.devices.NBIoTDevice
                                                   get_node_id()
                                                            method), 733
                                                    get_node_id()
                                                                        (digi.xbee.devices.Raw802Device
get_next_frame_id()
        (digi.xbee.devices.DigiPointDevice
                                          method),
                                                            method), 525
                                                    get_node_id() (digi.xbee.devices.RemoteDigiMeshDevice
get_next_frame_id() (digi.xbee.devices.IPDevice
                                                            method), 825
        method), 657
                                                    get_node_id() (digi.xbee.devices.RemoteDigiPointDevice
get_next_frame_id()
                                                            method), 843
        (digi.xbee.devices.LPWANDevice
                                          method),
                                                   get_node_id() (digi.xbee.devices.RemoteRaw802Device
        708
                                                            method), 806
get_next_frame_id()
                                                   get_node_id() (digi.xbee.devices.RemoteXBeeDevice
        (digi.xbee.devices.NBIoTDevice
                                          method),
                                                            method), 789
                                                    get_node_id() (digi.xbee.devices.RemoteZigBeeDevice
get_next_frame_id()
                                                            method), 863
                                                   get_node_id()
        (digi.xbee.devices.Raw802Device
                                                                           (digi.xbee.devices.WiFiDevice
                                          method),
```

*method*), 765 (digi.xbee.devices.CellularDevice get\_pan\_id() (digi.xbee.devices.XBeeDevice get\_node\_id() method), 683 get\_pan\_id() method), 502 (digi.xbee.devices.DigiMeshDevice get\_node\_id() (digi.xbee.devices.ZigBeeDevice method), 558 method), 627 get\_pan\_id() (digi.xbee.devices.DigiPointDevice get nowait() (digi.xbee.reader.XBeeQueue method), method), 589 get\_pan\_id() (digi.xbee.devices.IPDevice method), get\_nt\_limits() (digi.xbee.devices.DigiMeshNetwork 647 class method), 915 get\_pan\_id() (digi.xbee.devices.LPWANDevice get\_nt\_limits() (digi.xbee.devices.DigiPointNetwork method), 708 class method), 927 get\_pan\_id() (digi.xbee.devices.NBIoTDevice get\_nt\_limits() (digi.xbee.devices.Raw802Network method), 733 class method), 904 get\_pan\_id() (digi.xbee.devices.Raw802Device get\_nt\_limits() (digi.xbee.devices.XBeeNetwork method), 525 class method), 882 get\_pan\_id() (digi.xbee.devices.RemoteDigiMeshDevice get\_nt\_limits() (digi.xbee.devices.ZigBeeNetwork method), 825 class method), 893 get\_pan\_id() (digi.xbee.devices.RemoteDigiPointDevice get\_number\_devices() method), 843 (digi.xbee.devices.DigiMeshNetwork method), get\_pan\_id() (digi.xbee.devices.RemoteRaw802Device method), 806 get\_number\_devices() get\_pan\_id() (digi.xbee.devices.RemoteXBeeDevice (digi.xbee.devices.DigiPointNetwork method), method), 789 927 get\_pan\_id() (digi.xbee.devices.RemoteZigBeeDevice get\_number\_devices() method), 863 (digi.xbee.devices.Raw802Network method), get\_pan\_id() (digi.xbee.devices.WiFiDevice method), 765 get\_number\_devices() (digi.xbee.devices.XBeeDevice get\_pan\_id() (digi.xbee.devices.XBeeNetwork method), method), 502 (digi.xbee.devices.ZigBeeDevice get\_pan\_id() get\_number\_devices() method), 627 (digi.xbee.devices.ZigBeeNetwork method), get\_parameter() (digi.xbee.devices.AbstractXBeeDevice 893 method), 467 get\_ota\_max\_block\_size() get\_parameter() (digi.xbee.devices.CellularDevice (digi.xbee.devices.RemoteDigiMeshDevice method), 683 *method*), 825 get\_parameter() (digi.xbee.devices.DigiMeshDevice get\_ota\_max\_block\_size() method), 558 (digi.xbee.devices.RemoteDigiPointDevice get\_parameter() (digi.xbee.devices.DigiPointDevice *method*), 843 method), 590 get\_parameter() get\_ota\_max\_block\_size() (digi.xbee.devices.IPDevice (digi.xbee.devices.RemoteRaw802Device method), 657 get\_parameter() (digi.xbee.devices.LPWANDevice method), 806 get\_ota\_max\_block\_size() method), 708 (digi.xbee.devices.RemoteXBeeDevice get\_parameter() (digi.xbee.devices.NBIoTDevice method), 782 method), 733 get\_ota\_max\_block\_size() get\_parameter() (digi.xbee.devices.Raw802Device (digi.xbee.devices.RemoteZigBeeDevice method), 526 *method*), 863 get\_parameter() (digi.xbee.devices.RemoteDigiMeshDevice get\_packet\_received\_callbacks() method), 825 (digi.xbee.reader.PacketListener method), get\_parameter() (digi.xbee.devices.RemoteDigiPointDevice method), 843 get\_packet\_received\_from\_callbacks() get\_parameter() (digi.xbee.devices.RemoteRaw802Device (digi.xbee.reader.PacketListener method), 999 method), 807 get\_pan\_id() (digi.xbee.devices.AbstractXBeeDevice get\_parameter() (digi.xbee.devices.RemoteXBeeDevice *method*), 474 method), 781

```
get_parameter() (digi.xbee.devices.RemoteZigBeeDevice
                                                            method), 864
        method), 863
                                                   get_power_level()
                                                                          (digi.xbee.devices.WiFiDevice
get_parameter()
                       (digi.xbee.devices.WiFiDevice
                                                            method), 766
        method), 765
                                                   get_power_level() (digi.xbee.devices.XBeeDevice
get_parameter()
                       (digi.xbee.devices.XBeeDevice
                                                            method), 503
                                                   get power level()
        method), 486
                     (digi.xbee.devices.ZigBeeDevice
                                                            (digi.xbee.devices.ZigBeeDevice
get_parameter()
                                                                                             method),
        method), 627
                                                            627
get_parameter_string()
                                                   get_protocol() (digi.xbee.devices.AbstractXBeeDevice
        (digi.xbee.models.atcomm.ATCommand
                                                            method), 471
                                                   get_protocol() (digi.xbee.devices.CellularDevice
        method), 130
get_phone_number_byte_array()
                                                            method), 670
        (digi.xbee.packets.cellular.RXSMSPacket
                                                   get_protocol() (digi.xbee.devices.DigiMeshDevice
        method), 248
                                                            method), 542
get_phone_number_byte_array()
                                                   get_protocol() (digi.xbee.devices.DigiPointDevice
        (digi.xbee.packets.cellular.TXSMSPacket
                                                            method), 574
        method), 252
                                                   get_protocol()
                                                                             (digi.xbee.devices.IPDevice
get_power_level()
                                                            method), 657
        (digi.xbee.devices.AbstractXBeeDevice
                                                                        (digi.xbee.devices.LPWANDevice
                                                   get_protocol()
        method), 475
                                                            method), 708
get_power_level()
                                                   get_protocol()
                                                                         (digi.xbee.devices.NBIoTDevice
        (digi.xbee.devices.CellularDevice
                                         method),
                                                            method), 722
        673
                                                                        (digi.xbee.devices.Raw802Device
                                                   get_protocol()
get_power_level()
                                                            method), 512
        (digi.xbee.devices.DigiMeshDevice
                                         method),
                                                   get_protocol() (digi.xbee.devices.RemoteDigiMeshDevice
        558
                                                            method), 817
get_power_level()
                                                   get_protocol() (digi.xbee.devices.RemoteDigiPointDevice
        (digi.xbee.devices.DigiPointDevice
                                         method),
                                                            method), 836
                                                   get_protocol() (digi.xbee.devices.RemoteRaw802Device
get_power_level()
                         (digi.xbee.devices.IPDevice
                                                            method), 799
        method), 657
                                                   get_protocol() (digi.xbee.devices.RemoteXBeeDevice
get_power_level()
                                                            method), 790
        (digi.xbee.devices.LPWANDevice
                                         method),
                                                   get_protocol() (digi.xbee.devices.RemoteZigBeeDevice
                                                            method), 854
                                                   get_protocol()
get_power_level()
                                                                           (digi.xbee.devices.WiFiDevice
        (digi.xbee.devices.NBIoTDevice
                                         method),
                                                            method), 748
        733
                                                   get_protocol()
                                                                          (digi.xbee.devices.XBeeDevice
get_power_level()
                                                            method), 503
        (digi.xbee.devices.Raw802Device
                                                   get_protocol()
                                                                         (digi.xbee.devices.ZigBeeDevice
                                         method),
        526
                                                            method), 606
get_power_level()
                                                   get_pwm_duty_cycle()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                            (digi.xbee.devices.AbstractXBeeDevice
        method), 825
                                                            method), 478
get_power_level()
                                                   get_pwm_duty_cycle()
                                                            (digi.xbee.devices.CellularDevice
        (digi.xbee.devices.RemoteDigiPointDevice
                                                                                             method),
                                                            683
        method), 843
get_power_level()
                                                   get_pwm_duty_cycle()
        (digi.xbee.devices.RemoteRaw802Device
                                                            (digi.xbee.devices.DigiMeshDevice
                                                                                             method),
        method), 807
                                                            559
get_power_level()
                                                   get_pwm_duty_cycle()
        (digi.xbee.devices.RemoteXBeeDevice
                                                            (digi.xbee.devices.DigiPointDevice
                                                                                             method),
        method), 789
                                                            590
get_power_level()
                                                   get_pwm_duty_cycle()
        (digi.xbee.devices.RemoteZigBeeDevice
                                                            (digi.xbee.devices.IPDevice method), 658
```

```
get_pwm_duty_cycle()
                                                   get_role() (digi.xbee.devices.RemoteDigiPointDevice
                                          method),
        (digi.xbee.devices.LPWANDevice
                                                            method), 844
                                                   get_role() (digi.xbee.devices.RemoteRaw802Device
get_pwm_duty_cycle()
                                                            method), 808
        (digi.xbee.devices.NBIoTDevice
                                          method),
                                                   get_role()
                                                                    (digi.xbee.devices.RemoteXBeeDevice
                                                            method), 790
                                                                   (digi.xbee.devices.RemoteZigBeeDevice
get_pwm_duty_cycle()
                                                   get_role()
        (digi.xbee.devices.Raw802Device
                                          method),
                                                            method), 864
        526
                                                   get_role() (digi.xbee.devices.WiFiDevice method),
get_pwm_duty_cycle()
                                                            766
                                                   get_role() (digi.xbee.devices.XBeeDevice method),
        (digi.xbee.devices.RemoteDigiMeshDevice
        method), 826
                                                            504
get_pwm_duty_cycle()
                                                   get_role() (digi.xbee.devices.ZigBeeDevice method),
        (digi.xbee.devices.RemoteDigiPointDevice
                                                            628
        method), 844
                                                    get_root() (digi.xbee.fileSystem.FileSystemManager
get_pwm_duty_cycle()
                                                            method), 939
        (digi.xbee.devices.RemoteRaw802Device
                                                   get_route_info_callbacks()
        method), 807
                                                            (digi.xbee.reader.PacketListener
                                                                                              method),
get_pwm_duty_cycle()
        (digi.xbee.devices.RemoteXBeeDevice
                                                   get_route_record_received_callbacks()
        method), 790
                                                            (digi.xbee.reader.PacketListener method), 1000
get_pwm_duty_cycle()
                                                   get_route_table()
        (digi.xbee.devices.RemoteZigBeeDevice
                                                            (digi.xbee.models.zdo.RouteTableReader
        method), 864
                                                            method), 230
get_pwm_duty_cycle()
                                                    get_route_to_node()
        (digi.xbee.devices.WiFiDevice method), 766
                                                            (digi.xbee.devices.CellularDevice
                                                                                              method),
get_pwm_duty_cycle()
        (digi.xbee.devices.XBeeDevice
                                          method),
                                                   get_route_to_node()
        503
                                                            (digi.xbee.devices.DigiMeshDevice
                                                                                              method),
get_pwm_duty_cycle()
                                                            559
        (digi.xbee.devices.ZigBeeDevice
                                          method),
                                                   get_route_to_node()
        628
                                                            (digi.xbee.devices.DigiPointDevice
                                                                                              method),
get_queue()
                     (digi.xbee.reader.PacketListener
                                                            591
        method), 993
                                                   get_route_to_node() (digi.xbee.devices.IPDevice
get_read_timeout()
                                                            method), 658
        (digi.xbee.serial.XBeeSerialPort
                                          method).
                                                   get_route_to_node()
        1007
                                                            (digi.xbee.devices.LPWANDevice
                                                                                              method),
get_role()
               (digi.xbee.devices.AbstractXBeeDevice
                                                            709
                                                    get_route_to_node()
        method), 472
                    (digi.xbee.devices.CellularDevice
                                                            (digi.xbee.devices.NBIoTDevice
get_role()
                                                                                              method),
        method), 683
                                                            734
                   (digi.xbee.devices.DigiMeshDevice
                                                   get_route_to_node()
get_role()
                                                            (digi.xbee.devices.Raw802Device
        method), 559
                                                                                              method),
                   (digi.xbee.devices.DigiPointDevice
                                                            527
get_role()
        method), 591
                                                    get_route_to_node()
get_role() (digi.xbee.devices.IPDevice method), 658
                                                            (digi.xbee.devices.WiFiDevice method), 767
                    (digi.xbee.devices.LPWANDevice
get_role()
                                                   get_route_to_node()
        method), 709
                                                            (digi.xbee.devices.XBeeDevice
                                                                                              method),
get_role() (digi.xbee.devices.NBIoTDevice method),
                                                            495
        734
                                                    get_route_to_node()
get_role()
                    (digi.xbee.devices.Raw802Device
                                                            (digi.xbee.devices.ZigBeeDevice
                                                                                              method),
        method), 527
get_role() (digi.xbee.devices.RemoteDigiMeshDevice get_routes() (digi.xbee.devices.RemoteZigBeeDevice
                                                            method), 855
        method), 826
```

```
(digi.xbee.devices.ZigBeeDevice get_sync_ops_timeout()
get_routes()
        method), 614
                                                           (digi.xbee.devices.DigiMeshDevice
                                                                                           method).
get_serial_port()
                                                           560
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                  get_sync_ops_timeout()
        method), 826
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                           method),
get serial port()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                  get_sync_ops_timeout()
        method), 844
                                                           (digi.xbee.devices.IPDevice method), 659
get_serial_port()
                                                  get_sync_ops_timeout()
        (digi.xbee.devices.RemoteRaw802Device
                                                           (digi.xbee.devices.LPWANDevice
                                                                                           method),
        method), 808
                                                           710
get_serial_port()
                                                  get_sync_ops_timeout()
        (digi.xbee.devices.RemoteXBeeDevice
                                                           (digi.xbee.devices.NBIoTDevice
                                                                                           method),
        method), 782
                                                           735
get_serial_port()
                                                  get_sync_ops_timeout()
        (digi.xbee.devices.RemoteZigBeeDevice
                                                           (digi.xbee.devices.Raw802Device
                                                                                           method),
        method), 865
                                                           528
get_setting_default_value()
                                                  get_sync_ops_timeout()
        (digi.xbee.profile.XBeeProfile method), 968
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
get_sms_received_callbacks()
                                                           method), 827
        (digi.xbee.reader.PacketListener
                                         method),
                                                  get_sync_ops_timeout()
                                                           (digi.xbee.devices.RemoteDigiPointDevice
get_sock_info() (digi.xbee.xsocket.socket method),
                                                           method), 845
                                                  get_sync_ops_timeout()
                                                           (digi.xbee.devices.RemoteRaw802Device
get_socket_data_received_callbacks()
        (digi.xbee.reader.PacketListener method), 1000
                                                           method), 808
get_socket_data_received_from_callbacks(get_sync_ops_timeout()
        (digi.xbee.reader.PacketListener method), 1000
                                                           (digi.xbee.devices.RemoteXBeeDevice
get_socket_info()
                                                           method), 791
        (digi.xbee.devices.CellularDevice
                                         method),
                                                  get_sync_ops_timeout()
        672
                                                           (digi.xbee.devices.RemoteZigBeeDevice
get_socket_info()
                                                           method), 865
        (digi.xbee.devices.LPWANDevice
                                                  get_sync_ops_timeout()
                                         method),
                                                           (digi.xbee.devices.WiFiDevice method), 767
get_socket_info()
                                                  get sync ops timeout()
        (digi.xbee.devices.NBIoTDevice
                                         method).
                                                           (digi.xbee.devices.XBeeDevice
                                                                                           method),
                                                           504
get_socket_state_received_callbacks()
                                                  get_sync_ops_timeout()
        (digi.xbee.reader.PacketListener method), 1000
                                                           (digi.xbee.devices.ZigBeeDevice
                                                                                           method),
                                                           629
get_sockets_list()
                                                  get_usage_information()
        (digi.xbee.devices.CellularDevice
                                         method),
        671
                                                           (digi.xbee.filesystem.LocalXBeeFileSystemManager
get_sockets_list()
                                                           method), 953
        (digi.xbee.devices.LPWANDevice
                                         method),
                                                  get_user_data_relay_received_callbacks()
                                                           (digi.xbee.reader.PacketListener method), 999
get_sockets_list()
                                                  get_volume_info()
                                                           (digi.xbee.filesystem.FileSystemManager
        (digi.xbee.devices.NBIoTDevice
                                         method),
        735
                                                           method), 943
get_sync_ops_timeout()
                                                  get_wifi_ai_status()
        (digi.xbee.devices.AbstractXBeeDevice
                                                           (digi.xbee.devices.WiFiDevice method), 748
        method), 473
                                                  get_xbee_device_callbacks()
                                                           (digi.xbee.devices.CellularDevice
get_sync_ops_timeout()
                                                                                           method),
        (digi.xbee.devices.CellularDevice
                                         method),
                                                           684
        684
                                                  get xbee device callbacks()
```

(digi.xbee.devices.DigiMeshDevice 560	method),	has_digital_value() method),961	(digi.xbee.io.IOSample
<pre>get_xbee_device_callbacks()     (digi.xbee.devices.DigiPointDevice</pre>	method),	has_digital_values()  method), 961	(digi.xbee.io.IOSample
591 get_xbee_device_callbacks() (digi.xbee.devices.IPDevice method),	659	has_explicit_packets(	
get_xbee_device_callbacks() (digi.xbee.devices.LPWANDevice 710	method),	has_explicit_packets(	
get_xbee_device_callbacks() (digi.xbee.devices.NBIoTDevice 735	method),	has_explicit_packets(	
<pre>get_xbee_device_callbacks()      (digi.xbee.devices.Raw802Device</pre>	method),	has_explicit_packets(	Device method), 659
<pre>get_xbee_device_callbacks()       (digi.xbee.devices.WiFiDevice method</pre>	d), 767	(digi.xbee.devices.LP 710	
<pre>get_xbee_device_callbacks()     (digi.xbee.devices.XBeeDevice</pre>	method),	has_explicit_packets( (digi.xbee.devices.NB 736	
<pre>get_xbee_device_callbacks()      (digi.xbee.devices.ZigBeeDevice</pre>	method),	has_explicit_packets(	
getblocking() (digi.xbee.xsocket.socket	method),		) FiDevice method), 767
GetPathIdCmdRequest (class digi.xbee.models.filesystem), 170		has_explicit_packets(	)
GetPathIdCmdResponse (class digi.xbee.models.filesystem), 172 getsocketopt() (digi.xbee.xsocket.socket 1013	in method),	490 has_explicit_packets(     (digi.xbee.devices.Zig	
	method),	has_filesystem (digi.xbe tribute), 969	e.profile.XBeeProfile at-
Н		has_firmware_files (dig attribute), 969	i.xbee.profile.XBeeProfile
hardware_version (digi.xbee.profile.XBee tribute), 970 HardwareVersion (class in digi.xbee.ma	J	has_local_filesystem (digi.xbee.profile.XBe 969	eProfile attribute),
187 has_analog_value() (digi.xbee.io.		has_local_firmware_fi (digi.xbee.profile.XBe	
method), 961 has_analog_values() (digi.xbee.io.	IOSample		ee.devices.CellularDevice
<pre>method), 961 has_devices() (digi.xbee.devices.DigiMes method), 916</pre>	shNetwork		e.devices.DigiMeshDevice
has_devices() (digi.xbee.devices.DigiPoin method), 927	ntNetwork		e.devices.DigiPointDevice
has_devices() (digi.xbee.devices.Raw80 method), 904	)2Network	has_packets() (digi.xbee.d	levices.IPDevice method),
has_devices() (digi.xbee.devices.XBe method), 876	eeNetwork		ee.devices.LPWANDevice
has_devices() (digi.xbee.devices.ZigBe method), 893	eeNetwork		bee.devices.NBIoTDevice

has_packets() (digi.xbee.devices.Raw802Device method), 528	index() (digi.xbee.reader.ExplicitDataReceived method), 980
has_packets() (digi.xbee.devices.WiFiDevice method), 767	index() (digi.xbee.reader.FileSystemFrameReceived method), 992
has_packets() (digi.xbee.devices.XBeeDevice method), 490	<pre>index() (digi.xbee.reader.InitDiscoveryScan method),</pre>
has_packets() (digi.xbee.devices.ZigBeeDevice method), 629	<pre>index() (digi.xbee.reader.IOSampleReceived method),</pre>
has_power_supply_value() (digi.xbee.io.IOSample method), 961	index() (digi.xbee.reader.IPDataReceived method), 981
has_pwm_capability() (digi.xbee.io.IOLine method), 958	index() (digi.xbee.reader.MicroPythonDataReceived method), 984
has_remote_filesystem (digi.xbee.profile.XBeeProfile attribute),	index() (digi.xbee.reader.ModemStatusReceived method), 975
969 has_remote_firmware_files	index() (digi.xbee.reader.NetworkModified method), 977
(digi.xbee.profile.XBeeProfile attribute), 969	index() (digi.xbee.reader.PacketReceived method), 973
HashFileCmdRequest (class in digi.xbee.models.filesystem), 152	index() (digi.xbee.reader.PacketReceivedFrom method), 974
HashFileCmdResponse (class in digi.xbee.models.filesystem), 154	index() (digi.xbee.reader.RelayDataReceived method), 982
hex_string_to_bytes() (in module digi.xbee.util.utils), 458	index() (digi.xbee.reader.RouteInformationReceived method), 989
hex_to_string() (in module digi.xbee.util.utils), 461 here (digi.xbee.gelets sighes CreateSourcePoutePacket	index() (digi.xbee.reader.RouteReceived method), 989 index() (digi.xbee.reader.RouteRecordIndicatorReceived
hops (digi.xbee.packets.zigbee.CreateSourceRoutePacket attribute), 449	method), 987 index() (digi.xbee.reader.SMSReceived method), 981
hops (digi.xbee.packets.zigbee.RouteRecordIndicatorPack attribute), 445	
I	<pre>index() (digi.xbee.reader.SocketDataReceivedFrom</pre>
I2C_FUNCTIONALITY (digi.xbee.io.IOMode attribute), 962	index() (digi.xbee.reader.SocketStateReceived method), 985
id (digi.xbee.models.protocol.Role attribute), 213 id (digi.xbee.models.zdo.NeighborRelationship at-	index() (digi.xbee.reader.XBeeEvent method), 972 INDIRECT_TRANSMISSION
tribute), 234 id (digi.xbee.models.zdo.RouteStatus attribute), 231	(digi.xbee.models.options.TransmitOptions attribute), 206
ident (digi.xbee.reader.PacketListener attribute), 1000 index (digi.xbee.io.IOLine attribute), 958	InitDiscoveryScan (class in digi.xbee.reader), 990 insert() (digi.xbee.reader.BluetoothDataReceived
index (digi.xbee.profile.FirmwareBaudrate attribute),	method), 983
963 index (digi.xbee.profile.FirmwareParity attribute), 964	<pre>insert() (digi.xbee.reader.DataReceived method),</pre>
index (digi.xbee.profile.FirmwareStopbits attribute), 964	
701	insert () (digi.xbee.reader.DeviceDiscovered method), 978
index() (digi.xbee.reader.BluetoothDataReceived method), 983	978 insert() (digi.xbee.reader.DiscoveryProcessFinished method), 979
<pre>index() (digi.xbee.reader.BluetoothDataReceived</pre>	978 insert() (digi.xbee.reader.DiscoveryProcessFinished method), 979 insert() (digi.xbee.reader.EndDiscoveryScan
index() (digi.xbee.reader.BluetoothDataReceived method), 983	978 insert() (digi.xbee.reader.DiscoveryProcessFinished method), 979
<pre>index() (digi.xbee.reader.BluetoothDataReceived</pre>	978 insert() (digi.xbee.reader.DiscoveryProcessFinished method), 979 insert() (digi.xbee.reader.EndDiscoveryScan method), 991

	method), 990	IOValue (class in digi.xbee.io), 958
insert	() (digi.xbee.reader.IOSampleReceived method), 976	<pre>ip_addr (digi.xbee.models.message.IPMessage at- tribute), 201</pre>
insert	() (digi.xbee.reader.IPDataReceived method), 981	<pre>ip_protocol (digi.xbee.packets.network.RXIPv4Packe</pre>
insert	() (digi.xbee.reader.MicroPythonDataReceived method), 984	<pre>ip_protocol (digi.xbee.packets.network.TXIPv4Packet attribute), 343</pre>
insert	() (digi.xbee.reader.ModemStatusReceived method), 975	IPAddressingMode (class in digi.xbee.models.mode), 194
insert	() (digi.xbee.reader.NetworkModified method), 977	IPDataReceived (class in digi.xbee.reader), 980 IPDevice (class in digi.xbee.devices), 642
insert	() (digi.xbee.reader.PacketReceived method), 973	IPMessage (class in digi.xbee.models.message), 201 IPProtocol (class in digi.xbee.models.protocol), 212
insert	() (digi.xbee.reader.PacketReceivedFrom method), 974	<pre>is_alive() (digi.xbee.reader.PacketListener method),</pre>
insert	() (digi.xbee.reader.RelayDataReceived	is_apply_changes_enabled()
	method), 982	(digi.xbee.devices.AbstractXBeeDevice
insert	() (digi.xbee.reader.RouteInformationReceived	method), 473
insert	method), 989 () (digi.xbee.reader.RouteReceived method), 989	is_apply_changes_enabled() (digi.xbee.devices.CellularDevice method), 685
insert	() (digi.xbee.reader.RouteRecordIndicatorReceive	
	method), 988	(digi.xbee.devices.DigiMeshDevice method),
insert	() (digi.xbee.reader.SMSReceived method), 982	560
insert	() (digi.xbee.reader.SocketDataReceived	is_apply_changes_enabled()
	method), 986	(digi.xbee.devices.DigiPointDevice method),
insert		592
	method), 987	is_apply_changes_enabled()
insert	. 9	(digi.xbee.devices.IPDevice method), 659
	method), 985	is_apply_changes_enabled()
	() (digi.xbee.reader.XBeeEvent method), 972 _ascii() (in module digi.xbee.util.utils), 460	(digi.xbee.devices.LPWANDevice method), 711
	_bytes() (in module digi.xbee.util.utils), 458	is_apply_changes_enabled()
	_length() (in module digi.xbee.util.utils),	(digi.xbee.devices.NBIoTDevice method),
±110_00	460	736
Invali	dConfigurationException, 935	is_apply_changes_enabled()
Invali	dOperatingModeException, 935 dPacketException, 935	(digi.xbee.devices.Raw802Device method), 528
	ple(digi.xbee.packets.common.IODataSampleRx	: <i>IndicatopPacket</i> hanges_enabled()
	attribute), 288	(digi.xbee.devices.RemoteDigiMeshDevice
io_sam	ple (digi.xbee.packets.raw.RX16IOPacket at-	method), 827
	tribute), 367	is_apply_changes_enabled()
io_sam	ple (digi.xbee.packets.raw.RX64IOPacket attribute), 363	(digi.xbee.devices.RemoteDigiPointDevice method), 845
io_sam	ple(digi.xbee.packets.wifi.IODataSampleRxIndia	
	attribute), 427	(digi.xbee.devices.RemoteRaw802Device
IOData	SampleRxIndicatorPacket (class in	method), 808
TOD - 1-	digi.xbee.packets.common), 286	is_apply_changes_enabled()
TODATA	SampleRxIndicatorWifiPacket (class in digi.xbee.packets.wifi), 425	(digi.xbee.devices.RemoteXBeeDevice method), 791
TOLina	(class in digi.xbee.io), 957	is_apply_changes_enabled()
	(class in digi.xbee.io), 962	(digi.xbee.devices.RemoteZigBeeDevice
	le (class in digi.xbee.io), 958	method), 865
	leReceived (class in digi.xbee.reader), 976	is_apply_changes_enabled()

(digi.xbee.devices.WiFiDevice method), 768	method), 311
is_apply_changes_enabled()	$\verb is_broadcast ()  \textit{(digi.xbee.packets.devicecloud.SendDataRequestPackets) } \\$
(digi.xbee.devices.XBeeDevice method),	method), 314
504	$\verb is_broadcast()  (digi.xbee.packets.devicecloud.SendDataResponsePackets.devicecloud.SendDataRespo$
is_apply_changes_enabled()	method), 316
(digi.xbee.devices.ZigBeeDevice method), 629	<pre>is_broadcast() (digi.xbee.packets.digimesh.RouteInformationPacket</pre>
<pre>is_bit_enabled() (in module digi.xbee.util.utils),</pre>	<pre>is_broadcast() (digi.xbee.packets.filesystem.FSRequestPacket     method), 325</pre>
is_broadcast (digi.xbee.models.message.ExplicitXBeeMattribute), 200	Messageoadcast () (digi.xbee.packets.filesystem.FSResponsePacket method), 328
	ris_broadcast() (digi.xbee.packets.filesystem.RemoteFSRequestPacket method), 332
	Packebroadcast () (digi.xbee.packets.filesystem.RemoteFSResponsePackets.filesystem.filesy
	ePacketcoadcast () (digi.xbee.packets.network.RXIPv4Packet method), 339
**	ets_broadcast() (digi.xbee.packets.network.TXIPv4Packet method), 342
is_broadcast() (digi.xbee.packets.cellular.RXSMSPackets.dellular.d	kes_broadcast() (digi.xbee.packets.raw.RX16IOPacket method), 366
is_broadcast() (digi.xbee.packets.cellular.TXSMSPack method), 253	kes_broadcast() (digi.xbee.packets.raw.RX16Packet method), 358
	Packelproadcast () (digi.xbee.packets.raw.RX64IOPacket method), 361
is_broadcast() (digi.xbee.packets.common.ATCommQ method), 259	
is_broadcast()(digi.xbee.packets.common.ATCommR method), 262	ResponsePacketst() (digi.xbee.packets.raw.TX16Packet method), 349
is_broadcast() (digi.xbee.packets.common.ExplicitAd method), 293	ldræssingPackeast () (digi.xbee.packets.raw.TX64Packet method), 346
is_broadcast()(digi.xbee.packets.common.ExplicitRX method), 298	KluslidatorPackæst () (digi.xbee.packets.raw.TXStatusPacket method), 352
	mpsleRxIndidatosRackddigi.xbee.packets.relay.UserDataRelayOutputPacket method), 373
	utus <u>Packet</u> adcast () (digi.xbee.packets.relay.UserDataRelayPacket method), 370
is_broadcast() (digi.xbee.packets.common.ReceivePamethod), 264	ckst_broadcast() (digi.xbee.packets.socket.SocketBindListenPacket method), 409
	GanninandPackest () (digi.xbee.packets.socket.SocketClosePacket method), 396
	Gannana Response Palalieti.xbee.packets.socket.SocketCloseResponsePacket method), 399
	dicketbroadcast () (digi.xbee.packets.socket.SocketConnectPacket method), 390
	tatu <u>s Packet</u> dcast () (digi.xbee.packets.socket.SocketConnectResponsePack method), 393
	Respuest Backetast () (digi.xbee.packets.socket.SocketCreatePacket method), 376
	Resplonse Backetst () (digi.xbee.packets.socket.SocketCreateResponsePacket

*method*), 379

method), 412

is\_broadcast() (digi.xbee.packets.devicecloud.DeviceResponseStatusBacket(digi.xbee.packets.socket.SocketListenResponsePacket

is\_broadcast() (digi.xbee.packets.devicecloud.FrameErrorPacketlcast() (digi.xbee.packets.socket.SocketNewIPv4ClientPacket

method), 305

method), 308

```
method), 415
                                                             592
is_broadcast() (digi.xbee.packets.socket.SocketOptionRequestPacket_info_complete()
        method), 382
                                                             (digi.xbee.devices.IPDevice method), 643
is_broadcast() (digi.xbee.packets.socket.SocketOptionRespiberxeiPacketinfo_complete()
        method), 386
                                                             (digi.xbee.devices.LPWANDevice
                                                                                              method),
is broadcast()(digi.xbee.packets.socket.SocketReceiveFromPackets
                                                    is device info complete()
        method), 421
is_broadcast() (digi.xbee.packets.socket.SocketReceivePacket (digi.xbee.devices.NBIoTDevice
                                                                                              method),
        method), 417
is_broadcast() (digi.xbee.packets.socket.SocketSendPacketdevice_info_complete()
        method), 402
                                                             (digi.xbee.devices.Raw802Device
                                                                                              method),
is_broadcast() (digi.xbee.packets.socket.SocketSendToPacket
        method), 406
                                                    is_device_info_complete()
is_broadcast() (digi.xbee.packets.socket.SocketStatePacket
                                                             (digi.xbee.devices.RemoteDigiMeshDevice
        method), 424
                                                             method), 827
is_broadcast() (digi.xbee.packets.wifi.IODataSampleRxxIndicatorWeifiPacket_complete()
                                                             (digi.xbee.devices.RemoteDigiPointDevice
        method), 428
is_broadcast() (digi.xbee.packets.wifi.RemoteATCommandResponsedWiff)PackEet
                                                    is_device_info_complete()
        method), 435
is broadcast() (digi.xbee.packets.wifi.RemoteATCommandWifiPatikatxbee.devices.RemoteRaw802Device
        method), 432
                                                             method), 808
is_broadcast() (digi.xbee.packets.zigbee.CreateSourceRoutePacket_info_complete()
                                                             (digi.xbee.devices.RemoteXBeeDevice
        method), 450
is broadcast() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatahBdok&D1
                                                    is device info complete()
        method), 453
is_broadcast() (digi.xbee.packets.zigbee.RegisterDeviceStatusPudkgi.xbee.devices.RemoteZigBeeDevice
        method), 443
                                                             method), 854
is_broadcast() (digi.xbee.packets.zigbee.RegisterJoining_Devise:Packetnfo_complete()
        method), 440
                                                             (digi.xbee.devices.WiFiDevice method), 768
is_broadcast() (digi.xbee.packets.zigbee.RouteRecordInsligatorPacketinfo_complete()
        method), 445
                                                             (digi.xbee.devices.XBeeDevice
                                                                                              method),
is_connected(digi.xbee.filesystem.LocalXBeeFileSystemManager504
        attribute), 951
                                                    is_device_info_complete()
is_connected (digi.xbee.xsocket.socket attribute),
                                                             (digi.xbee.devices.ZigBeeDevice
                                                                                              method),
        1014
                    (digi.xbee.devices.CellularDevice
                                                    is dir
                                                              (digi.xbee.filesystem.FileSystemElement
is connected()
        method), 670
                                                             tribute), 937
is_connected()
                    (digi.xbee.devices.LPWANDevice
                                                    IS_DIR (digi.xbee.models.options.DirResponseFlag at-
        method), 711
                                                             tribute), 211
is_connected()
                      (digi.xbee.devices.NBIoTDevice
                                                    is_discovery_running()
                                                             (digi.xbee.devices.DigiMeshNetwork method),
        method), 736
is_connected()
                       (digi.xbee.devices.WiFiDevice
                                                             916
                                                    is_discovery_running()
        method), 751
                                                             (digi.xbee.devices.DigiPointNetwork method),
is_device_info_complete()
        (digi.xbee.devices.AbstractXBeeDevice
        method), 470
                                                    is_discovery_running()
is_device_info_complete()
                                                             (digi.xbee.devices.Raw802Network
                                                                                              method),
        (digi.xbee.devices.CellularDevice
                                                             904
                                          method),
        670
                                                    is_discovery_running()
                                                             (digi.xbee.devices.XBeeNetwork
is_device_info_complete()
                                                                                              method),
        (digi.xbee.devices.DigiMeshDevice
                                                             876
                                          method),
                                                    is_discovery_running()
is_device_info_complete()
                                                             (digi.xbee.devices.ZigBeeNetwork
                                                                                              method),
        (digi.xbee.devices.DigiPointDevice
                                          method),
                                                             893
```

is_interface_open	768
(digi.xbee.comm_interface.XBeeCommunicationl attribute), 464	Interface() (digi.xbee.devices.XBeeDevice method), 493
is_interface_open	is_open() (digi.xbee.devices.ZigBeeDevice method),
(digi.xbee.serial.XBeeSerialPort attribute),	630
1006	is_open() (digi.xbee.profile.XBeeProfile method), 968
is_known_node_addr()	is_remote() (digi.xbee.devices.AbstractXBeeDevice
(digi.xbee.models.address.XBee16BitAddress class method), 196	method), 473 is_remote() (digi.xbee.devices.CellularDevice
is_known_node_addr()	method), 685
(digi.x bee.models. address. XBee 64Bit Address	is_remote() (digi.xbee.devices.DigiMeshDevice
class method), 198	method), 561
is_last (digi.xbee.models.filesystem.OpenDirCmdRespo attribute), 161	mise_remote() (digi.xbee.devices.DigiPointDevice method), 592
is_last (digi.xbee.models.filesystem.ReadDirCmdRespo. attribute), 169	
IS_LAST (digi.xbee.models.options.DirResponseFlag	is_remote() (digi.xbee.devices.LPWANDevice
attribute), 211	method), 711
is_low_memory (digi.xbee.models.zdo.Route at- tribute), 232	is_remote() (digi.xbee.devices.NBIoTDevice method), 737
is_many_to_one (digi.xbee.models.zdo.Route at- tribute), 232	is_remote() (digi.xbee.devices.Raw802Device method), 529
is_node_in_network()	$\verb is_remote()  (\textit{digi.xbee.devices.RemoteDigiMeshDevice} $
(digi.xbee.devices.DigiMeshNetwork method), 916	<pre>method), 827 is_remote() (digi.xbee.devices.RemoteDigiPointDevice</pre>
is_node_in_network()	method), 845
(digi.xbee.devices.DigiPointNetwork method), 927	is_remote() (digi.xbee.devices.RemoteRaw802Device method), 808
is_node_in_network()	is_remote() (digi.xbee.devices.RemoteXBeeDevice
(digi.xbee.devices.Raw802Network method), 905	method), 781 is_remote() (digi.xbee.devices.RemoteZigBeeDevice
is_node_in_network()	method), 865
(digi.xbee.devices.XBeeNetwork method), 883	is_remote() (digi.xbee.devices.WiFiDevice method), 768
is_node_in_network()	<pre>is_remote() (digi.xbee.devices.XBeeDevice method),</pre>
(digi.xbee.devices.ZigBeeNetwork method),	493
893 is_op_mode_valid()	is_remote() (digi.xbee.devices.ZigBeeDevice method), 630
(digi.xbee.sender.PacketSender method),	is_route_record_required
1004	(digi.xbee.models.zdo.Route attribute), 232
<pre>is_open() (digi.xbee.devices.CellularDevice method),</pre>	is_rssi (digi.xbee.devices.LinkQuality attribute), 932
685	is_running() (digi.xbee.reader.PacketListener
is_open() (digi.xbee.devices.DigiMeshDevice method), 561	method), 993 is_secure (digi.xbee.filesystem.FileSystemElement at-
is_open() (digi.xbee.devices.DigiPointDevice	tribute), 937
method), 592	IS_SECURE (digi.xbee.models.options.DirResponseFlag
is_open() (digi.xbee.devices.IPDevice method), 659	attribute), 211
is_open() (digi.xbee.devices.LPWANDevice method),	is_valid() (digi.xbee.models.address.XBee16BitAddress
711 is_open() (digi.xbee.devices.NBIoTDevice method),	<pre>class method), 196 is_valid() (digi.xbee.models.address.XBee64BitAddress</pre>
736	class method), 197
is_open() (digi.xbee.devices.Raw802Device method), 528	<pre>is_valid() (digi.xbee.models.address.XBeeIMEIAddress</pre>
is_open() (digi.xbee.devices.WiFiDevice method),	isAlive() (digi.xbee.reader.PacketListener method),

1000	lq (digi.xbee.devices.LinkQuality attribute), 932
J	lq (digi.xbee.models.zdo.Neighbor attribute), 235 lq_a2b (digi.xbee.devices.Connection attribute), 933
join() (digi.xbee.reader.PacketListener method), 1000	1q_b2a (digi.xbee.devices.Connection attribute), 933
<pre>join() (digi.xbee.reader.XBeeQueue method), 1003</pre>	M
Key (digi.xbee.packets.zigbee.RegisterJoiningDevicePacke	mac_capabilities (digi.xbee.models.zdo.NodeDescriptor attribute), 229
attribute), 439	make_directory() (digi.xbee.filesystem.FileSystemManager method), 939
L	make_directory()(digi.xbee.filesystem.LocalXBeeFileSystemManage
LegacyHardwareVersion (class in	method), 951
digi.xbee.models.hw), 189	manufacturer_code
length (digi.xbee.packets.digimesh.RouteInformationPacattribute), 320	cket (digi.xbee.models.zdo.NodeDescriptor at- tribute), 229
<pre>length_to_int() (in module digi.xbee.util.utils),      459</pre>	<pre>max_buffer_size(digi.xbee.models.zdo.NodeDescriptor</pre>
letter (digi.xbee.models.hw.LegacyHardwareVersion	max_in_transfer_size
attribute), 190	(digi.xbee.models.zdo.NodeDescriptor at-
LinkQuality (class in digi.xbee.devices), 931	tribute), 229
<pre>list_directory() (digi.xbee.fileSystem.FileSystemMo</pre>	(digi.xbee.models.zdo.NodeDescriptor at-
$\verb list_directory()  \textit{(digi.xbee.filesystem.LocalXBeeFilesystem.LocalXBeeFilesystem.}  \textit{(a)} \textit{(digi.xbee.filesystem.LocalXBeeFilesystem.LocalXBeeFilesystem.LocalXBeeFilesystem.}  \textit{(a)} (digi.xbee.filesystem.LocalXBeeFil$	leSystemMdridgets), 229
method), 951	MAX_TIME_BETWEEN_REQUESTS (digi.xbee.devices.XBeeNetwork attribute),
listen() (digi.xbee.xsocket.socket method), 1010	1_7
local_interface(digi.xbee.models.message.UserDatatribute), 203	MAX_TIME_BETWEEN_SCANS
local_port (digi.xbee.models.info.SocketInfo at- tribute), 191	(digi.xbee.devices.XBeeNetwork attribute), 874
LocalXBeeFileSystemManager (class in digi.xbee.filesystem), 950	MicroPythonDataReceived (class in digi.xbee.reader), 984
log (digi.xbee.devices.AbstractXBeeDevice attribute), 484	min_io_sample_payload() (digi.xbee.io.IOSample static method), 960
log (digi.xbee.devices.CellularDevice attribute), 685	MIN_TIME_BETWEEN_REQUESTS
log (digi.xbee.devices.DigiMeshDevice attribute), 561 log (digi.xbee.devices.DigiPointDevice attribute), 592	(digi.xbee.devices.XBeeNetwork attribute), 874
log (digi.xbee.devices.IPDevice attribute), 660	MIN_TIME_BETWEEN_SCANS
log (digi.xbee.devices.LPWANDevice attribute), 711	(digi.xbee.devices.XBeeNetwork attribute),
log (digi.xbee.devices.NBIoTDevice attribute), 737	874
log (digi.xbee.devices.Raw802Device attribute), 529	<pre>modem_status(digi.xbee.packets.common.ModemStatusPacket</pre>
log (digi.xbee.devices.RemoteDigiMeshDevice at-	attribute), 284
tribute), 827	ModemStatus (class in digi.xbee.models.status), 216
log (digi.xbee.devices.RemoteDigiPointDevice at-	ModemStatusPacket (class in
tribute), 845	digi.xbee.packets.common), 283
log (digi.xbee.devices.RemoteRaw802Device attribute), 809	ModemStatusReceived (class in digi.xbee.reader), 975
log (digi.xbee.devices.RemoteXBeeDevice attribute), 791	move() (digi.xbee.filesystem.FileSystemManager method), 943
log (digi.xbee.devices.RemoteZigBeeDevice attribute), 865	move_element() (digi.xbee.filesystem.LocalXBeeFileSystemManager method), 952
log (digi.xbee.devices.WiFiDevice attribute), 768	N
log (digi.xbee.devices.XBeeDevice attribute), 504	N
log (digi.xbee.devices.ZigBeeDevice attribute), 630 LPWANDevice (class in digi.xbee.devices), 696	name (digi.xbee.filesystem.FileSystemElement attribute), 937

needs\_id() (digi.xbee.packets.devicecloud.DeviceResponsePacket

needs\_id() (digi.xbee.packets.devicecloud.FrameErrorPacket

needs\_id() (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket

 ${\tt needs\_id}$  () (digi.xbee.packets.devicecloud.SendDataResponsePacket

needs\_id() (digi.xbee.packets.digimesh.RouteInformationPacket

needs\_id() (digi.xbee.packets.filesystem.RemoteFSRequestPacket

(digi.xbee.packets.raw.RX16IOPacket

(digi.xbee.packets.raw.RX64IOPacket

(digi.xbee.packets.raw.RX16Packet

(digi.xbee.packets.raw.RX64Packet

needs\_id() (digi.xbee.packets.filesystem.FSRequestPacket

needs\_id() (digi.xbee.packets.filesystem.FSResponsePacket

name (digi.xbee.models.filesystem.CreateDirCmdRequest needs\_id() (digi.xbee.packets.common.RemoteATCommandPacket attribute), 157 method), 268 name (digi.xbee.models.filesystem.DeleteCmdRequest atneeds id() (digi.xbee.packets.common.RemoteATCommandResponsePa

method), 300

method), 304

method), 307

method), 310

*method*), 313

method), 318

method), 320

method), 324

method), 327

method), 330

method), 335

method), 337

method), 341

method), 366

method), 357

method), 361

method), 354

needs id()

needs id()

- method), 272 tribute), 178
- needs\_id() (digi.xbee.packets.common.TransmitPacket name (digi.xbee.models.filesystem.FileIdNameCmd attribute), 137 method), 277
- name (digi.xbee.models.filesystem.GetPathIdCmdRequest needs id()(digi.xbee.packets.common.TransmitStatusPacket attribute), 171 method), 281
- name (digi.xbee.models.filesystem.HashFileCmdRequest needs\_id() (digi.xbee.packets.devicecloud.DeviceRequestPacket attribute), 153
- name (digi.xbee.models.filesystem.OpenDirCmdRequest attribute), 160
- name (digi.xbee.models.filesystem.OpenFileCmdRequest attribute), 139
- (digi.xbee.models.filesystem.RenameCmdRequestname attribute), 175
- name (digi.xbee.models.filesystem.VolFormatCmdRequest needs\_id() (digi.xbee.packets.devicecloud.SendDataRequestPacket attribute), 184
- (digi.xbee.models.filesystem.VolStatCmdRequestname attribute), 180
- (digi.xbee.profile.XBeeProfileSetting attribute), name
- name (digi.xbee.reader.PacketListener attribute), 1001 NBIoTDevice (class in digi.xbee.devices), 722
- ND\_PACKET\_FINISH (digi.xbee.devices.XBeeNetwork
- attribute), 873 ND\_PACKET\_REMOTE (digi.xbee.devices.XBeeNetwork
- attribute), 873
- needs\_id() (digi.xbee.packets.base.GenericXBeePacket needs\_id() (digi.xbee.packets.filesystem.RemoteFSResponsePacket *method*), 243
- method), 247 needs\_id() (digi.xbee.packets.base.XBeeAPIPacket needs\_id() (digi.xbee.packets.network.TXIPv4Packet
- method), 241
- needs id() (digi.xbee.packets.cellular.RXSMSPacket method), 248
- needs id() (digi.xbee.packets.cellular.TXSMSPacket method), 251
- needs\_id() (digi.xbee.packets.common.ATCommPacket needs\_id() method), 255
- needs id()(digi.xbee.packets.common.ATCommQueuePackeds id() *method*), 258
- needs id() (digi.xbee.packets.common.ATCommResponsePacket id() (digi.xbee.packets.raw.TX16Packet method), 261 method), 347
- needs\_id() (digi.xbee.packets.common.ExplicitAddressingPacket\_id() (digi.xbee.packets.raw.TX64Packet *method*), 293 method), 344

needs\_id() (digi.xbee.packets.base.UnknownXBeePacketneeds\_id() (digi.xbee.packets.network.RXIPv4Packet

- needs\_id()(digi.xbee.packets.common.ExplicitRXIndicator\*Packet\*d() (digi.xbee.packets.raw.TXStatusPacket *method*), 298 method), 351
- needs\_id() (digi.xbee.packets.common.IODataSampleRxIndiatoirRacketdigi.xbee.packets.relay.UserDataRelayOutputPacket method), 287 method), 371
- needs\_id() (digi.xbee.packets.common.ModemStatusPacketeds\_id() (digi.xbee.packets.relay.UserDataRelayPacket method), 284 method), 368
- needs\_id() (digi.xbee.packets.common.ReceivePacket needs\_id() (digi.xbee.packets.socket.SocketBindListenPacket method), 264 method), 408

```
needs_id() (digi.xbee.packets.socket.SocketClosePacket NetworkDiscoveryStatus
                                                                                            (class
                                                                                                         in
         method), 395
                                                               digi.xbee.models.status), 222
needs id() (digi.xbee.packets.socket.SocketCloseResponNeRacketkEventReason (class in digi.xbee.devices),
        method), 398
                                                               931
needs_id() (digi.xbee.packets.socket.SocketConnectPackitetworkEventType (class in digi.xbee.devices), 930
        method), 388
                                                      NetworkModified (class in digi.xbee.reader), 977
needs id() (digi.xbee.packets.socket.SocketConnectResponse Packet (digi.xbee.models.filesystem.RenameCmdRequest
         method), 392
                                                               attribute), 174
needs id() (digi.xbee.packets.socket.SocketCreatePacketnext hop (digi.xbee.models.zdo.Route attribute), 232
         method), 375
                                                      node (digi.xbee.models.zdo.Neighbor attribute), 234
needs_id() (digi.xbee.packets.socket.SocketCreateResponsePacket(digi.xbee.devices.Connection attribute), 932
         method), 378
                                                      node_b (digi.xbee.devices.Connection attribute), 933
needs_id() (digi.xbee.packets.socket.SocketListenResponseBacketscriptor (class in digi.xbee.models.zdo), 228
        method), 411
                                                      NodeDescriptorReader
                                                                                           (class
                                                                                                         in
needs_id() (digi.xbee.packets.socket.SocketNewIPv4ClientPacket digi.xbee.models.zdo), 227
         method), 414
                                                      NONE
                                                               (digi.xbee.models.options.ReceiveOptions
needs_id() (digi.xbee.packets.socket.SocketOptionRequestPacket tribute), 204
         method), 381
                                                      NONE (digi.xbee.models.options.RemoteATCmdOptions
needs_id() (digi.xbee.packets.socket.SocketOptionResponsePacketattribute), 207
        method), 384
                                                      NONE
                                                              (digi.xbee.models.options.TransmitOptions
needs_id() (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                               tribute), 205
        method), 422
                                                      np_value (digi.xbee.filesystem.FileSystemManager at-
needs_id() (digi.xbee.packets.socket.SocketReceivePacket
                                                               tribute), 939
         method), 418
                                                      number of hops (digi.xbee.packets.zigbee.CreateSourceRoutePacket
needs_id() (digi.xbee.packets.socket.SocketSendPacket
                                                               attribute), 449
                                                      number_of_hops(digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
        method), 401
needs_id()(digi.xbee.packets.socket.SocketSendToPacket
                                                               attribute), 445
        method), 404
needs_id() (digi.xbee.packets.socket.SocketStatePacket O
         method), 425
                                                      offset (digi.xbee.models.filesystem.ReadFileCmdRequest
needs_id() (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacketsute), 145
         method), 426
                                                      offset (digi.xbee.models.filesystem.ReadFileCmdResponse)
needs_id() (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket), 147
        method), 436
                                                      offset (digi.xbee.models.filesystem.WriteFileCmdRequest
needs_id() (digi.xbee.packets.wifi.RemoteATCommandWifiPacket attribute), 149
        method), 430
                                                                   (digi.xbee.packets.base.GenericXBeePacket
                                                      op mode
needs_id() (digi.xbee.packets.zigbee.CreateSourceRoutePacket
                                                               attribute), 244
        method), 448
                                                      op_mode (digi.xbee.packets.base.UnknownXBeePacket
needs_id() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacketbute), 247
        method), 454
                                                      op_mode
                                                                       (digi.xbee.packets.base.XBeeAPIPacket
needs_id() (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket attribute), 242
        method), 441
                                                      op_mode (digi.xbee.packets.base.XBeePacket attribute),
needs_id() (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket238
        method), 438
                                                      op_mode (digi.xbee.packets.cellular.RXSMSPacket at-
needs_id() (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket_tribute), 250
         method), 444
                                                      op_mode (digi.xbee.packets.cellular.TXSMSPacket at-
Neighbor (class in digi.xbee.models.zdo), 234
                                                               tribute), 253
NeighborDiscoveryMode
                                     (class
                                                                   (digi.xbee.packets.common.ATCommPacket
                                                      op_mode
         digi.xbee.models.mode), 194
                                                                attribute), 256
NeighborFinder (class in digi.xbee.models.zdo), 235
                                                      op_mode (digi.xbee.packets.common.ATCommQueuePacket
NeighborRelationship
                                    (class
                                                  in
                                                                attribute), 259
        digi.xbee.models.zdo), 233
                                                      op_mode (digi.xbee.packets.common.ATCommResponsePacket
NeighborTableReader
                                   (class
                                                  in
                                                                attribute), 263
         digi.xbee.models.zdo), 232
```

- op\_mode (digi.xbee.packets.common.ExplicitAddressingPacket\_mode (digi.xbee.packets.raw.TX64Packet attribute), attribute), 293
- op\_mode(digi.xbee.packets.common.ExplicitRXIndicatorPacketnode (digi.xbee.packets.raw.TXStatusPacket atattribute), 297 tribute), 352
- op\_mode (digi.xbee.packets.common.IODataSampleRxIndioatonRawka(digi.xbee.packets.relay.UserDataRelayOutputPacket attribute), 289 attribute), 373
- op\_mode (digi.xbee.packets.common.ModemStatusPacket op\_mode (digi.xbee.packets.relay.UserDataRelayPacket attribute), 370 attribute), 285
- op\_mode (digi.xbee.packets.common.ReceivePacket at- op\_mode (digi.xbee.packets.socket.SocketBindListenPacket attribute), 409 tribute), 266
- $op\_mode(digi.xbee.packets.common.RemoteATCommandPapokettode)$ (digi.xbee.packets.socket.SocketClosePacket attribute), 270 attribute), 396
- op\_mode (digi.xbee.packets.common.RemoteATCommandResponsePacketi.xbee.packets.socket.SocketCloseResponsePacket attribute), 274 attribute), 399
- op\_mode (digi.xbee.packets.common.TransmitPacket at- op\_mode (digi.xbee.packets.socket.SocketConnectPacket *tribute*), 279 attribute), 390
- $op\_mode(digi.xbee.packets.common.TransmitStatusPacketop\_mode(digi.xbee.packets.socket.SocketConnectResponsePacketop\_mode(digi.xbee.packets.socket.SocketConnectResponsePacketop\_mode(digi.xbee.packets.socket.SocketConnectResponsePacketop\_mode(digi.xbee.packets.socket.Socket.$ attribute), 282 attribute), 393
- op\_mode (digi.xbee.packets.devicecloud.DeviceRequestPacket\_mode (digi.xbee.packets.socket.SocketCreatePacket attribute), 302 attribute), 376
- $op\_mode(digi.xbee.packets.devicecloud.DeviceResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacketmode(digi.xbee.packetmode(digi.xb$ attribute), 305 attribute), 379
- op\_mode (digi.xbee.packets.devicecloud.DeviceResponseStatusRacket(digi.xbee.packets.socket.SocketListenResponsePacket attribute), 308 attribute), 412
- op mode (digi.xbee.packets.devicecloud.FrameErrorPacketp mode (digi.xbee.packets.socket.SocketNewIPv4ClientPacket attribute), 311 attribute), 415
- op\_mode (digi.xbee.packets.devicecloud.SendDataRequestBacketode (digi.xbee.packets.socket.SocketOptionRequestPacket attribute), 315 attribute), 383
- op\_mode (digi.xbee.packets.devicecloud.SendDataResponsePacketde (digi.xbee.packets.socket.SocketOptionResponsePacket attribute), 317 attribute), 386
- op\_mode(digi.xbee.packets.digimesh.RouteInformationPacket\_mode(digi.xbee.packets.socket.SocketReceiveFromPacket attribute), 321 attribute), 421
- op\_mode (digi.xbee.packets.filesystem.FSRequestPacket op\_mode (digi.xbee.packets.socket.SocketReceivePacket attribute), 325 attribute), 418
- op\_mode (digi.xbee.packets.filesystem.FSResponsePacket op\_mode (digi.xbee.packets.socket.SocketSendPacket atattribute), 328 tribute), 402
- op\_mode(digi.xbee.packets.filesystem.RemoteFSRequestPacketmode (digi.xbee.packets.socket.SocketSendToPacket attribute), 332 attribute), 406
- op\_mode(digi.xbee.packets.filesystem.RemoteFSResponseRapketode (digi.xbee.packets.socket.SocketStatePacket attribute), 334 attribute), 424
- op mode (digi.xbee.packets.network.RXIPv4Packet at*tribute*), 339
- op\_mode (digi.xbee.packets.network.TXIPv4Packet attribute), 342
- (digi.xbee.packets.raw.RX16IOPacket atop\_mode tribute), 365
- op\_mode (digi.xbee.packets.raw.RX16Packet attribute), 359
- op\_mode (digi.xbee.packets.raw.RX64IOPacket attribute), 363
- op\_mode (digi.xbee.packets.raw.RX64Packet attribute), 356
- 349

- op mode (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacket attribute), 429
- attribute), 435
- $op\_mode(digi.xbee.packets.wifi.RemoteATCommandWifiPacket$ attribute), 432
- op\_mode (digi.xbee.packets.zigbee.CreateSourceRoutePacket attribute), 451
- $op\_mode(digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket$ attribute), 453
- $op\_mode(digi.xbee.packets.zigbee.RegisterDeviceStatusPacket$ attribute), 443
- op\_mode (digi.xbee.packets.raw.TX16Packet attribute), op\_mode (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket attribute), 440

op\_mode (digi.xbee.packets.zigbee.RouteRecordIndicatorPacketLon\_data (digi.xbee.packets.socket.SocketOptionResponsePacket attribute), 447 attribute), 385 open () (digi.xbee.comm\_interface.XBeeCommunicationInterface.xBee.models.filesystem.OpenFileCmdRequest method), 464 attribute), 139 open () (digi.xbee.devices.CellularDevice method), 670 options (digi.xbee.packets.devicecloud.SendDataRequestPacket open () (digi.xbee.devices.DigiMeshDevice method), attribute), 313 options (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket open() (digi.xbee.devices.DigiPointDevice method), attribute), 438 574 OPTIONS\_CLOSE\_SOCKET open () (digi.xbee.devices.IPDevice method), 660 (digi.xbee.packets.network.TXIPv4Packet open () (digi.xbee.devices.LPWANDevice method), 711 attribute), 340 open () (digi.xbee.devices.NBIoTDevice method), 722 OPTIONS\_LEAVE\_SOCKET\_OPEN open () (digi.xbee.devices.Raw802Device method), 512 (digi.xbee.packets.network.TXIPv4Packet open () (digi.xbee.devices.WiFiDevice method), 748 attribute), 340 open () (digi.xbee.devices.XBeeDevice method), 485 OTAFirmwareUpdateStatusPacket open () (digi.xbee.devices.ZigBeeDevice method), 605 digi.xbee.packets.zigbee), 451 open () (digi.xbee.profile.XBeeProfile method), 967 output () (digi.xbee.models.filesystem.CloseDirCmdRequest open () (digi.xbee.serial.XBeeSerialPort method), 1008 method), 164 OpenDirCmdRequest output() (digi.xbee.models.filesystem.CloseDirCmdResponse (class in digi.xbee.models.filesystem), 159 method), 165 OpenDirCmdResponse (class output() (digi.xbee.models.filesystem.CloseFileCmdRequest in digi.xbee.models.filesystem), 161 method), 143 OpenFileCmdRequest output() (digi.xbee.models.filesystem.CloseFileCmdResponse (class in digi.xbee.models.filesystem), 138 method), 144 OpenFileCmdResponse output () (digi.xbee.models.filesystem.CreateDirCmdRequest (class in digi.xbee.models.filesystem), 140 method), 157 operating\_mode (digi.xbee.devices.CellularDevice output() (digi.xbee.models.filesystem.CreateDirCmdResponse method), 158 attribute), 685 operating\_mode (digi.xbee.devices.DigiMeshDevice output () (digi.xbee.models.filesystem.DeleteCmdRequest attribute), 561 method), 178 operating\_mode (digi.xbee.devices.DigiPointDevice output () (digi.xbee.models.filesystem.DeleteCmdResponse attribute), 593 method), 179 operating\_mode (digi.xbee.devices.IPDevice output() (digi.xbee.models.filesystem.FileIdCmd *tribute*), 660 method), 136 operating mode (digi.xbee.devices.LPWANDevice output () (digi.xbee.models.filesystem.FileIdNameCmd method), 137 attribute), 712 operating\_mode (digi.xbee.devices.NBIoTDevice atoutput() (digi.xbee.models.filesystem.FSCmd tribute), 737 method), 133 operating\_mode (digi.xbee.devices.Raw802Device output () (digi.xbee.models.filesystem.GetPathIdCmdRequest attribute), 529 method), 171 operating mode (digi.xbee.devices.WiFiDevice atoutput () (digi.xbee.models.filesystem.GetPathIdCmdResponse tribute), 768 method), 173 output () (digi.xbee.models.filesystem.HashFileCmdRequest operating\_mode (digi.xbee.devices.XBeeDevice attribute), 486 method), 154  $\verb"output" () \textit{ (digi.xbee.models.filesystem.HashFileCmdResponse}$ operating\_mode (digi.xbee.devices.ZigBeeDevice attribute), 630 method), 155 OperatingMode (class in digi.xbee.models.mode), 192 output () (digi.xbee.models.filesystem.OpenDirCmdRequest OperationNotSupportedException, 935 *method*), 160 option (digi.xbee.packets.socket.SocketOptionRequestPackettput () (digi.xbee.models.filesystem.OpenDirCmdResponse attribute), 381 method), 162 option (digi.xbee.packets.socket.SocketOptionResponsePacketput () (digi.xbee.models.filesystem.OpenFileCmdRequest attribute), 385 method), 139

1076 Index

attribute), 381

option\_data(digi.xbee.packets.socket.SocketOptionRequestPacket) (digi.xbee.models.filesystem.OpenFileCmdResponse

method), 141

- output () (digi.xbee.models.filesystem.ReadDirCmdRequestitput () (digi.xbee.packets.common.RemoteATCommandPacket method), 167 method), 270
- output () (digi.xbee.models.filesystem.ReadDirCmdResponsePacket nethod), 169 (digi.xbee.packets.common.RemoteATCommandResponsePacket nethod), 275
- output() (digi.xbee.models.filesystem.ReadFileCmdRequesttput() (digi.xbee.packets.common.TransmitPacket method), 146 method), 279
- output () (digi.xbee.models.filesystem.ReadFileCmdResp@nsteput () (digi.xbee.packets.common.TransmitStatusPacket method), 148 method), 283
- output () (digi.xbee.models.filesystem.RenameCmdRequestutput () (digi.xbee.packets.devicecloud.DeviceRequestPacket method), 175 method), 302
- output () (digi.xbee.models.filesystem.RenameCmdResponsetput () (digi.xbee.packets.devicecloud.DeviceResponsePacket method), 176 method), 305
- output () (digi.xbee.models.filesystem.UnknownFSCmd output () (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket method), 134 method), 308
- output () (digi.xbee.models.filesystem.VolFormatCmdRequestput () (digi.xbee.packets.devicecloud.FrameErrorPacket method), 184 method), 311
- output () (digi.xbee.models.filesystem.VolFormatCmdResportsput () (digi.xbee.packets.devicecloud.SendDataRequestPacket method), 186 method), 315
- output () (digi.xbee.models.filesystem.VolStatCmdRequestoutput () (digi.xbee.packets.devicecloud.SendDataResponsePacket method), 181 method), 317
- output () (digi.xbee.models.filesystem.VolStatCmdResponsetput () (digi.xbee.packets.digimesh.RouteInformationPacket method), 183 method), 322
- output () (digi.xbee.models.filesystem.WriteFileCmdRequesttput () (digi.xbee.packets.filesystem.FSRequestPacket method), 150 method), 325
- output () (digi.xbee.models.filesystem.WriteFileCmdResponsePacket () (digi.xbee.packets.filesystem.FSResponsePacket method), 152 method), 328
- output() (digi.xbee.packets.base.GenericXBeePacket output() (digi.xbee.packets.filesystem.RemoteFSRequestPacket method), 244 method), 332
- output() (digi.xbee.packets.base.UnknownXBeePacket output() (digi.xbee.packets.filesystem.RemoteFSResponsePacket method), 247 method), 334
- output() (digi.xbee.packets.base.XBeeAPIPacket output() (digi.xbee.packets.network.RXIPv4Packet method), 242 method), 339
- output () (digi.xbee.packets.base.XBeePacket method), output () (digi.xbee.packets.network.TXIPv4Packet method), 342
- output() (digi.xbee.packets.cellular.RXSMSPacket output() (digi.xbee.packets.raw.RX16IOPacket method), 250 method), 365
- output() (digi.xbee.packets.cellular.TXSMSPacket output() (digi.xbee.packets.raw.RX16Packet method), method), 253 359
- output() (digi.xbee.packets.common.ATCommPacket output() (digi.xbee.packets.raw.RX64IOPacket method), 256 method), 363
- output () (digi.xbee.packets.common.ATCommQueuePackettput () (digi.xbee.packets.raw.RX64Packet method), method), 259 356
- output () (digi.xbee.packets.common.ATCommResponsePacketput () (digi.xbee.packets.raw.TX16Packet method), method), 263 349
- output () (digi.xbee.packets.common.ExplicitAddressingPacketput () (digi.xbee.packets.raw.TX64Packet method), method), 293 346
- output () (digi.xbee.packets.common.ExplicitRXIndicatorPackett () (digi.xbee.packets.raw.TXStatusPacket method), 297 method), 352
- output () (digi.xbee.packets.common.IODataSampleRxIndiattpvPacketdigi.xbee.packets.relay.UserDataRelayOutputPacket method), 289 method), 373
- output () (digi.xbee.packets.common.ModemStatusPacketoutput () (digi.xbee.packets.relay.UserDataRelayPacket method), 285 method), 370
- output() (digi.xbee.packets.common.ReceivePacket output() (digi.xbee.packets.socket.SocketBindListenPacket method), 266 method), 409

```
output () (digi.xbee.packets.socket.SocketClosePacket PacketReceived (class in digi.xbee.reader), 972
              method), 396
                                                                                    PacketReceivedFrom (class in digi.xbee.reader),
output () (digi.xbee.packets.socket.SocketCloseResponsePacket
             method), 399
                                                                                    Packet Sender (class in digi.xbee.sender), 1004
output () (digi.xbee.packets.socket.SocketConnectPacket parameter (digi.xbee.models.atcomm.ATCommand at-
             method), 390
                                                                                                  tribute), 130
output () (digi.xbee.packets.socket.SocketConnectResponsePacket*ter (digi.xbee.packets.common.ATCommPacket
              method), 393
                                                                                                  attribute), 255
output() (digi.xbee.packets.socket.SocketCreatePacket parameter(digi.xbee.packets.common.ATCommQueuePacket
              method), 376
                                                                                                  attribute), 258
\verb|output()| (\textit{digi.xbee.packets.socket.SocketCreateResponsePacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packets.common.RemoteATCommandPacketheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.packetheter(digi.xbee.pa
                                                                                                  attribute), 269
              method), 379
output () (digi.xbee.packets.socket.SocketListenResponsePacketmeter (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
             method), 412
                                                                                                  attribute), 431
output () (digi.xbee.packets.socket.SocketNewIPv4ClientPacketnt
                                                                                                    (digi.xbee.devices.RemoteZigBeeDevice at-
              method), 416
                                                                                                  tribute), 854
output () (digi.xbee.packets.socket.SocketOptionRequestPacketty
                                                                                                   (digi.xbee.profile.FirmwareParity attribute),
             method), 383
                                                                                                  964
output() (digi.xbee.packets.socket.SocketOptionResponsePacket_socket_list()
             method), 386
                                                                                                  (digi.xbee.models.info.SocketInfo
                                                                                                                                                             static
output() (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                                                                  method), 191
             method), 421
                                                                                    path (digi.xbee.filesystem.FileSystemElement attribute),
output () (digi.xbee.packets.socket.SocketReceivePacket
                                                                                                  937
                                                                                    path (digi.xbee.packets.devicecloud.SendDataRequestPacket
              method), 418
output() (digi.xbee.packets.socket.SocketSendPacket
                                                                                                  attribute), 313
             method), 403
                                                                                    PATTERN (digi.xbee.models.address.XBee16BitAddress
output () (digi.xbee.packets.socket.SocketSendToPacket
                                                                                                  attribute), 195
                                                                                    PATTERN (digi.xbee.models.address.XBee64BitAddress
             method), 406
output() (digi.xbee.packets.socket.SocketStatePacket
                                                                                                  attribute), 197
             method), 424
                                                                                    PATTERN (digi.xbee.models.address.XBeeIMEIAddress
output () (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacketribute), 198
              method), 429
                                                                                    PATTERN_PHONE_NUMBER
                                                                                                                                          (in
                                                                                                                                                          module
output () (digi.xbee.packets.wifi.RemoteATCommandResponseWifiRligketbee.packets.cellular), 247
                                                                                    payload(digi.xbee.packets.socket.SocketReceiveFromPacket
             method), 435
output () (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
                                                                                                  attribute), 422
                                                                                    payload (digi.xbee.packets.socket.SocketReceivePacket
             method), 432
output () (digi.xbee.packets.zigbee.CreateSourceRoutePacket
                                                                                                  attribute), 419
                                                                                    payload (digi.xbee.packets.socket.SocketSendPacket at-
             method), 451
output () (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacketbute), 401
                                                                                    payload (digi.xbee.packets.socket.SocketSendToPacket
              method), 453
output () (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
                                                                                                  attribute), 405
             method), 443
                                                                                    pclose_file() (digi.xbee.filesystem.FileSystemManager
output () (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket method), 947
             method), 440
                                                                                    pget_file_hash() (digi.xbee.fileSystem.FileSystemManager
output() (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket method), 949
              method), 447
                                                                                    pget_path_id() (digi.xbee.filesystem.FileSystemManager
                                                                                                  method), 944
Р
                                                                                    phone_number (digi.xbee.models.message.SMSMessage
                                                                                                  attribute), 202
packet (digi.xbee.sender.SyncRequestSender attribute),
                                                                                    phone_number (digi.xbee.packets.cellular.RXSMSPacket
              1005
                                                                                                  attribute), 248
PACKET_ACKNOWLEDGED
                                                                                    phone_number (digi.xbee.packets.cellular.TXSMSPacket
              (digi.xbee.models.options.ReceiveOptions
                                                                                                  attribute), 253
              attribute), 204
                                                                                    plist_directory()
PacketListener (class in digi.xbee.reader), 992
```

(digi.xbee.filesystem.FileSystemManager PowerLevel (class in digi.xbee.models.status), 218 *method*), 945 pread\_file() (digi.xbee.filesystem.FileSystemManager pmake directory() method), 948 (digi.xbee.filesystem.FileSystemManager prelease\_path\_id() *method*), 945 (digi.xbee.filesystem.FileSystemManager POINT MULTIPOINT MODE method), 950 (digi.xbee.models.options.ReceiveOptions (digi.xbee.filesystem.FileSystemManager premove() attribute), 204 method), 946 POINT\_MULTIPOINT\_MODE prename() (digi.xbee.filesystem.FileSystemManager (digi.xbee.models.options.TransmitOptions method), 949 attribute), 206 prepare\_for\_update() (digi.xbee.reader.BluetoothDataReceived (digi.xbee.firmware.UpdateConfigurer pop() method), 983 method), 955 pop () (digi.xbee.reader.DataReceived method), 974 prepare\_total (digi.xbee.firmware.UpdateConfigurer pop () (digi.xbee.reader.DeviceDiscovered method), 978 attribute), 954 (digi.xbee.reader.DiscoveryProcessFinished profile\_description\_file pop() method), 979 (digi.xbee.profile.XBeeProfile attribute), (digi.xbee.reader.EndDiscoveryScan method), 970 pop() profile\_file (digi.xbee.profile.XBeeProfile pop() (digi.xbee.reader.ExplicitDataReceived method), tribute), 968 profile\_id(digi.xbee.models.message.ExplicitXBeeMessage (digi.xbee.reader.FileSystemFrameReceived attribute), 200 pop() method), 992 profile\_id(digi.xbee.packets.common.ExplicitAddressingPacket pop () (digi.xbee.reader.InitDiscoveryScan method), 990 attribute), 294 (digi.xbee.reader.IOSampleReceived method), profile\_id(digi.xbee.packets.common.ExplicitRXIndicatorPacket attribute), 299 pop () (digi.xbee.reader.IPDataReceived method), 981 profile\_settings (digi.xbee.profile.XBeeProfile at-(digi.xbee.reader.MicroPythonDataReceived tribute), 970 pop() method), 984progress\_cb() (digi.xbee.firmware.UpdateConfigurer pop() (digi.xbee.reader.ModemStatusReceived method), method), 955 protocol (digi.xbee.models.info.SocketInfo attribute), pop () (digi.xbee.reader.NetworkModified method), 977 191 pop () (digi.xbee.reader.PacketReceived method), 973 protocol (digi.xbee.models.message.IPMessage atpop() (digi.xbee.reader.PacketReceivedFrom method), tribute), 202 protocol (digi.xbee.packets.socket.SocketCreatePacket (digi.xbee.reader.RelayDataReceived method), attribute), 375 pop() 982 protocol (digi.xbee.profile.XBeeProfile attribute), 971 (digi.xbee.reader.RouteInformationReceived purge\_port() (digi.xbee.serial.XBeeSerialPort pop() method), 989 method), 1007 pop () (digi.xbee.reader.RouteReceived method), 990 put () (digi.xbee.reader.XBeeQueue method), 1003 pop() (digi.xbee.reader.RouteRecordIndicatorReceived (digi.xbee.filesystem.FileSystemManager put dir() method), 942 method), 988 pop () (digi.xbee.reader.SMSReceived method), 982 put dir() (digi.xbee.filesystem.LocalXBeeFileSystemManager pop() (digi.xbee.reader.SocketDataReceived method), method), 952 put\_file() (digi.xbee.fileSystem.FileSystemManager (digi.xbee.reader.SocketDataReceivedFrom method), 942 pop() put\_file() (digi.xbee.filesystem.LocalXBeeFileSystemManager method), 987 (digi.xbee.reader.SocketStateReceived method), pop() method), 952 985 put\_nowait() (digi.xbee.reader.XBeeQueue method), pop () (digi.xbee.reader.XBeeEvent method), 972 1003 popen\_file() (digi.xbee.fileSystem.FileSystemManager PWM (digi.xbee.io.IOMode attribute), 962 method), 947 pwm\_at\_command (digi.xbee.io.IOLine attribute), 958 power\_supply\_value (digi.xbee.io.IOSample pwrite file() (digi.xbee.filesystem.FileSystemManager attribute), 961

Index 1079

method), 948

Q	read_data() (digi.xbee.devices.DigiPointDevice
qsize() (digi.xbee.reader.XBeeQueue method), 1003 quit_reading() (digi.xbee.comm_interface.XBeeCom	method), 593 เทนิทินิต <del>tioทิปักนิย</del> ศาลce <sup>(</sup> digi.xbee.devices.IPDevice method),
method), 464 quit_reading() (digi.xbee.serial.XBeeSerialPort	read_data() (digi.xbee.devices.LPWANDevice method), 712
method), 1007	read_data() (digi.xbee.devices.NBIoTDevice method), 737
Raw802Device (class in digi.xbee.devices), 512	read_data() (digi.xbee.devices.Raw802Device method), 529
Raw802Network (class in digi.xbee.devices), 896 reachable (digi.xbee.devices.AbstractXBeeDevice attribute), 484	read_data() (digi.xbee.devices.WiFiDevice method), 768
reachable (digi.xbee.devices.CellularDevice attribute), 685	read_data() (digi.xbee.devices.XBeeDevice method), 489
reachable (digi.xbee.devices.DigiMeshDevice attribute), 561	read_data() (digi.xbee.devices.ZigBeeDevice method), 630
reachable (digi.xbee.devices.DigiPointDevice attribute), 593	read_data_from() (digi.xbee.devices.CellularDevice method), 685
reachable (digi.xbee.devices.IPDevice attribute), 660 reachable (digi.xbee.devices.LPWANDevice at-	<pre>read_data_from() (digi.xbee.devices.DigiMeshDevice</pre>
tribute), 712 reachable (digi.xbee.devices.NBIoTDevice attribute),	<pre>read_data_from() (digi.xbee.devices.DigiPointDevice</pre>
737 reachable (digi.xbee.devices.Raw802Device at-	read_data_from() (digi.xbee.devices.IPDevice method), 647
tribute), 529 reachable (digi.xbee.devices.RemoteDigiMeshDevice	<pre>read_data_from() (digi.xbee.devices.LPWANDevice</pre>
attribute), 827	<pre>read_data_from() (digi.xbee.devices.NBIoTDevice</pre>
reachable (digi.xbee.devices.RemoteDigiPointDevice attribute), 845 reachable (digi.xbee.devices.RemoteRaw802Device	<pre>read_data_from() (digi.xbee.devices.Raw802Device</pre>
attribute), 809	read_data_from() (digi.xbee.devices.WiFiDevice method), 769
reachable (digi.xbee.devices.RemoteXBeeDevice at- tribute), 791	read_data_from() (digi.xbee.devices.XBeeDevice method), 489
reachable (digi.xbee.devices.RemoteZigBeeDevice at- tribute), 865	<pre>read_data_from() (digi.xbee.devices.ZigBeeDevice</pre>
reachable (digi.xbee.devices.WiFiDevice attribute), 768	<pre>read_device_info()      (digi.xbee.devices.AbstractXBeeDevice</pre>
reachable (digi.xbee.devices.XBeeDevice attribute), 505	<pre>method), 470 read_device_info()</pre>
reachable (digi.xbee.devices.ZigBeeDevice attribute), 630	(digi.xbee.devices.CellularDevice method), 685
READ (digi.xbee.models.options.FileOpenRequestOption attribute), 210	read_device_info()dPage_act (digi.xbee.devices.DigiMeshDevice method),
READ_AS_MANY (digi.xbee.models.filesystem.ReadFileCr attribute), 145	read_device_info()
read_byte() (digi.xbee.serial.XBeeSerialPort method), 1006	(digi.xbee.devices.DigiPointDevice method), 594
read_bytes() (digi.xbee.serial.XBeeSerialPort method), 1007	read_device_info() (digi.xbee.devices.IPDevice method), 660
read_data() (digi.xbee.devices.CellularDevice method), 685	read_device_info() (digi.xbee.devices.LPWANDevice method),
read_data() (digi.xbee.devices.DigiMeshDevice method), 561	712 read_device_info()

(digi.xbee.devices.NBIoTDevice method),	method), 594	
737 read_device_info()	read_io_sample()  method),661	(digi.xbee.devices.IPDevice
(digi.xbee.devices.Raw802Device method), 530		(digi.xbee.devices.LPWANDevice
<pre>read_device_info()      (digi.xbee.devices.RemoteDigiMeshDevice</pre>		(digi.xbee.devices.NBIoTDevice
method), 827 read_device_info()		(digi.xbee.devices.Raw802Device
(digi.xbee.devices.RemoteDigiPointDevice method), 845		(digi.xbee.devices. Remote DigiMesh Device
read_device_info() (digi.xbee.devices.RemoteRaw802Device		(digi.xbee.devices. Remote Digi Point Device
<pre>method), 809 read_device_info()</pre>	read_io_sample()  method), 809	(digi.xbee.devices.RemoteRaw802Device
(digi.xbee.devices.RemoteXBeeDevice method), 791	read_io_sample()  method), 792	(digi.xbee.devices.RemoteXBeeDevice
<pre>read_device_info()      (digi.xbee.devices.RemoteZigBeeDevice</pre>	read_io_sample()  method), 866	(digi.xbee.devices.RemoteZigBeeDevice
<pre>method), 865 read_device_info()</pre>	<pre>read_io_sample()     method), 769</pre>	
(digi.xbee.devices.WiFiDevice method), 769 read_device_info()	read_io_sample()  method), 505	(digi.xbee.devices.XBeeDevice
(digi.xbee.devices.XBeeDevice method), 505	<pre>read_io_sample()     method), 632</pre>	(digi.xbee.devices.ZigBeeDevice
<pre>read_device_info()      (digi.xbee.devices.ZigBeeDevice method), 631</pre>	method), 686	(digi.xbee.devices.CellularDevice
read_existing() (digi.xbee.serial.XBeeSerialPort	read_ip_data()  method), 645	(digi.xbee.devices.IPDevice
method), 1007	read_ip_data()	(digi.xbee.devices.LPWANDevice
read_expl_data() (digi.xbee.devices.DigiMeshDevice method), 543	read_ip_data()	(digi.xbee.devices.NBIoTDevice
read_expl_data() (digi.xbee.devices.DigiPointDevice method), 575	read_ip_data()	(digi.xbee.devices.WiFiDevice
read_expl_data() (digi.xbee.devices.ZigBeeDevice method), 608	method), 769 read_ip_data_fro	
read_expl_data_from() (digi.xbee.devices.DigiMeshDevice method),	687	vices.CellularDevice method),
544 read_expl_data_from()	read_ip_data_from method), 646	om() (digi.xbee.devices.IPDevice
(digi.xbee.devices.DigiPointDevice method), 576	read_ip_data_fro	om () vices.LPWANDevice method),
read_expl_data_from()	713	4
(digi.xbee.devices.ZigBeeDevice method), 609	_	om () vices.NBIoTDevice method),
read_file() (digi.xbee.filesystem.FileSystemManager method), 940	738 read_ip_data_fro	om ()
read_io_sample() (digi.xbee.devices.AbstractXBeeDe	evice (digi.xbee.dev	vices.WiFiDevice method), 770
method), 477	ReadDirCmdReques	
read_io_sample() (digi.xbee.devices.CellularDevice method), 686	ReadDirCmdRespon	lels.filesystem), 166 ase (class in
<pre>read_io_sample() (digi.xbee.devices.DigiMeshDevice</pre>	e digi.xbee.mod	lels.filesystem), 168
<pre>method), 563 read_io_sample() (digi.xbee.devices.DigiPointDevice</pre>	ReadFileCmdReque digi.xbee.mod	est (class in lels.filesystem), 145

```
ReadFileCmdResponse
                                  (class
                                                 in
                                                              attribute), 438
                                                                            (digi.xbee.models.zdo.Neighbor
        digi.xbee.models.filesystem), 147
                                                     relationship
                                                              attribute), 234
ReadProfileException, 967
real_status (digi.xbee.packets.common.ATCommResponsePacketaReceived (class in digi.xbee.reader), 982
        attribute), 261
                                                     remote address
                                                                           (digi.xbee.models.info.SocketInfo
real status (digi.xbee.packets.common.RemoteATCommandRespanseReel91
                                                     remote address (digi.xbee.packets.socket.SocketNewIPv4ClientPacket
        attribute), 273
receive_options (digi.xbee.packets.common.ExplicitRXIndicatorPrickate), 414
         attribute), 299
                                                     remote_device (digi.xbee.models.message.ExplicitXBeeMessage
receive_options (digi.xbee.packets.common.IODataSampleRxInaltivibatPacket
        attribute), 288
                                                     remote_device (digi.xbee.models.message.XBeeMessage
receive_options (digi.xbee.packets.common.ReceivePacket
                                                              attribute), 199
        attribute), 265
                                                     remote_file_system_image
receive_options (digi.xbee.packets.filesystem.RemoteFSResponselPgi:kbree.profile.XBeeProfile
                                                                                               attribute),
        attribute), 335
receive_options(digi.xbee.packets.raw.RX16IOPacketremote_port
                                                                           (digi.xbee.models.info.SocketInfo
         attribute), 367
                                                              attribute), 191
receive_options (digi.xbee.packets.raw.RX16Packet remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket
                                                              attribute), 414
        attribute), 358
                                                                                                       in
receive options (digi.xbee.packets.raw.RX64IOPackeRemoteATCmdOptions
                                                                                       (class
        attribute), 361
                                                              digi.xbee.models.options), 206
receive_options(digi.xbee.packets.raw.RX64Packet RemoteATCommandPacket
                                                                                         (class
                                                                                                       in
                                                              digi.xbee.packets.common), 267
         attribute), 354
receive options (digi.xbee.packets.wifi.IODataSampleRexbadieArtaCVWifitRadxdResponsePacket
                                                                                              (class
                                                                                                      in
         attribute), 427
                                                              digi.xbee.packets.common), 271
receive_options (digi.xbee.packets.zigbee.OTAFirmw@retUpdotA$1atusnRacketResponseWifiPacket (class
         attribute), 455
                                                              in digi.xbee.packets.wifi), 433
receive_options (digi.xbee.packets.zigbee.RouteRecordIndicaterPackethandWifiPacket
                                                                                            (class
                                                                                                       in
        attribute), 445
                                                              digi.xbee.packets.wifi), 429
ReceiveOptions (class in digi.xbee.models.options), RemoteDigiMeshDevice
                                                                                         (class
                                                                                                       in
                                                              digi.xbee.devices), 817
ReceivePacket (class in digi.xbee.packets.common),
                                                     RemoteDigiPointDevice
                                                                                         (class
                                                                                                       in
                                                              digi.xbee.devices), 835
                                                     RemoteFSRequestPacket
recover_device() (in module digi.xbee.recovery),
                                                                                         (class
                                                                                                       in
         1004
                                                              digi.xbee.packets.filesystem), 329
                                                                                          (class
RecoveryException, 936
                                                     RemoteFSResponsePacket
                                                                                                       in
recv() (digi.xbee.xsocket.socket method), 1011
                                                              digi.xbee.packets.filesystem), 332
recvfrom() (digi.xbee.xsocket.socket method), 1011
                                                     RemoteRaw802Device (class in digi.xbee.devices),
region_lock (digi.xbee.profile.XBeeProfile attribute),
                                                              798
                                                     RemoteXBeeDevice (class in digi.xbee.devices), 780
         970
register_joining_device()
                                                     RemoteZigBeeDevice (class in digi.xbee.devices),
         (digi.xbee.devices.ZigBeeDevice
                                           method),
                                                              853
                                                                    (digi.xbee.filesystem.FileSystemManager
                                                     remove()
register_joining_device_async()
                                                              method), 940
                                                                    (digi.xbee.reader.BluetoothDataReceived
        (digi.xbee.devices.ZigBeeDevice
                                           method),
                                                     remove()
        613
                                                              method), 983
RegisterDeviceStatusPacket
                                       (class
                                                 in
                                                     remove()
                                                                  (digi.xbee.reader.DataReceived method),
                                                              975
        digi.xbee.packets.zigbee), 440
                                                     \verb"remove" () \textit{ (digi.xbee.reader.DeviceDiscovered method)},
RegisterJoiningDevicePacket
                                        (class
                                                 in
        digi.xbee.packets.zigbee), 437
RegisterKeyOptions
                                  (class
                                                     remove() (digi.xbee.reader.DiscoveryProcessFinished
        digi.xbee.models.options), 209
                                                              method), 979
registrant_address
                                                                        (digi.xbee.reader.EndDiscoveryScan
                                                     remove()
         (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket
                                                              method), 991
```

```
(digi.xbee.reader.ExplicitDataReceived REPEATER_MODE (digi.xbee.models.options.TransmitOptions
remove()
         method), 980
                                                               attribute), 206
remove()
           (digi.xbee.reader.FileSystemFrameReceived
                                                      request data (digi.xbee.packets.devicecloud.DeviceRequestPacket
         method), 992
                                                               attribute), 301
remove()
                   (digi.xbee.reader.InitDiscoveryScan
                                                      request_data(digi.xbee.packets.devicecloud.DeviceResponsePacket
        method), 990
                                                               attribute), 304
                   (digi.xbee.reader.IOSampleReceived
                                                      request id(digi.xbee.packets.devicecloud.DeviceRequestPacket
remove()
         method), 976
                                                               attribute), 301
remove() (digi.xbee.reader.IPDataReceived method),
                                                      request_id(digi.xbee.packets.devicecloud.DeviceResponsePacket
                                                               attribute), 304
         981
remove() (digi.xbee.reader.MicroPythonDataReceived
                                                      reset()
                                                                      (digi.xbee.devices.AbstractXBeeDevice
         method), 984
                                                               method), 470
               (digi.xbee.reader.ModemStatusReceived
                                                                 (digi.xbee.devices.CellularDevice method),
                                                      reset()
remove()
         method), 975
                                                               687
remove() (digi.xbee.reader.NetworkModified method),
                                                      reset() (digi.xbee.devices.DigiMeshDevice method),
         977
remove() (digi.xbee.reader.PacketReceived method),
                                                      reset() (digi.xbee.devices.DigiPointDevice method),
         973
                                                               595
                 (digi.xbee.reader.PacketReceivedFrom
                                                      reset () (digi.xbee.devices.IPDevice method), 661
remove()
        method), 974
                                                                 (digi.xbee.devices.LPWANDevice method),
                  (digi.xbee.reader.RelayDataReceived
                                                               713
remove()
        method), 983
                                                      reset () (digi.xbee.devices.NBIoTDevice method), 739
remove() (digi.xbee.reader.RouteInformationReceived
                                                                 (digi.xbee.devices.Raw802Device method),
                                                      reset()
         method), 989
remove()
            (digi.xbee.reader.RouteReceived method),
                                                      reset()
                                                                   (digi.xbee.devices.RemoteDigiMeshDevice
                                                               method), 828
remove() (digi.xbee.reader.RouteRecordIndicatorReceiverteset()
                                                                   (digi.xbee.devices.RemoteDigiPointDevice
        method), 988
                                                               method), 846
remove() (digi.xbee.reader.SMSReceived method), 982
                                                                    (digi.xbee.devices.RemoteRaw802Device
                                                      reset()
                 (digi.xbee.reader.SocketDataReceived
                                                               method), 810
remove()
         method), 986
                                                      reset()
                                                                       (digi.xbee.devices.RemoteXBeeDevice
            (digi.xbee.reader.SocketDataReceivedFrom
                                                               method), 781
remove()
        method), 987
                                                                      (digi.xbee.devices.RemoteZigBeeDevice
                                                      reset()
                 (digi.xbee.reader.SocketStateReceived
                                                               method), 866
remove()
         method), 985
                                                      reset () (digi.xbee.devices.WiFiDevice method), 770
remove() (digi.xbee.reader.XBeeEvent method), 972
                                                      reset () (digi.xbee.devices.XBeeDevice method), 491
remove device() (digi.xbee.devices.DigiMeshNetworkreset() (digi.xbee.devices.ZigBeeDevice method), 632
        method), 916
                                                      reset_settings (digi.xbee.profile.XBeeProfile at-
remove_device() (digi.xbee.devices.DigiPointNetwork
                                                               tribute), 969
                                                      responder_addr (digi.xbee.packets.digimesh.RouteInformationPacket
        method), 928
remove device() (digi.xbee.devices.Raw802Network
                                                               attribute), 322
         method), 905
                                                      response (digi.xbee.models.atcomm.ATCommandResponse
remove device()
                      (digi.xbee.devices.XBeeNetwork
                                                               attribute), 130
        method), 884
                                                      restore_after_update()
remove_device() (digi.xbee.devices.ZigBeeNetwork
                                                               (digi.xbee.firmware.UpdateConfigurer
         method), 893
                                                               method), 955
remove_element() (digi.xbee.filesystem.LocalXBeeFileSystemManageral (digi.xbee.firmware.UpdateConfigurer
        method), 952
                                                               attribute), 954
RenameCmdRequest
                                 (class
                                                      reverse()
                                                                    (digi.xbee.reader.BluetoothDataReceived
        digi.xbee.models.filesystem), 173
                                                               method), 983
RenameCmdResponse
                                 (class
                                                                   (digi.xbee.reader.DataReceived method),
                                                  in
                                                      reverse()
         digi.xbee.models.filesystem), 175
REPEATER_MODE (digi.xbee.models.options.ReceiveOptionseverse()
                                                                         (digi.xbee.reader.DeviceDiscovered
         attribute), 204
                                                               method), 978
```

reverse() (digi.xbee.reader.DiscoveryProcessFinished	358
method), 979 reverse() (digi.xbee.reader.EndDiscoveryScan	rf_data (digi.xbee.packets.raw.RX64IOPacket at- tribute), 362
method), 991	rf_data (digi.xbee.packets.raw.RX64Packet attribute),
reverse() (digi.xbee.reader.ExplicitDataReceived	355
method), 980	rf_data (digi.xbee.packets.raw.TX16Packet attribute),
reverse() (digi.xbee.reader.FileSystemFrameReceived	348
method), 992	rf_data (digi.xbee.packets.raw.TX64Packet attribute),
reverse() (digi.xbee.reader.InitDiscoveryScan	345
method), 991	$\verb rf_data  (digi.x bee. packets. wifi. IOD at a Sample Rx Indicator Wifi Packet IOD at a Sample Rx Indicator Wifi Pac$
reverse() (digi.xbee.reader.IOSampleReceived	attribute), 427
method), 977	Role (class in digi.xbee.models.protocol), 213
reverse() (digi.xbee.reader.IPDataReceived method), 981	role (digi.xbee.models.zdo.NodeDescriptor attribute), 228
$\verb reverse ()  \textit{(digi.xbee.reader.MicroPythonDataReceived} $	Route (class in digi.xbee.models.zdo), 231
method), 984	route_cmd_options
reverse() (digi.xbee.reader.ModemStatusReceived method), 975	(digi.xbee.packets.zigbee.CreateSourceRoutePacket attribute), 449
reverse() (digi.xbee.reader.NetworkModified	RouteInformationPacket (class in
method), 977	digi.xbee.packets.digimesh), 318
reverse() (digi.xbee.reader.PacketReceived method), 973	RouteInformationReceived (class in digi.xbee.reader), 988
reverse() (digi.xbee.reader.PacketReceivedFrom	RouteReceived (class in digi.xbee.reader), 989
method), 974	RouteRecordIndicatorPacket (class in
${\tt reverse()} \qquad \textit{(digi.xbee.reader.RelayDataReceived)}$	digi.xbee.packets.zigbee), 443
method), 983	RouteRecordIndicatorReceived (class in
reverse() (digi.xbee.reader.RouteInformationReceived	digi.xbee.reader), 987
method), 989	RouteStatus (class in digi.xbee.models.zdo), 231
reverse() (digi.xbee.reader.RouteReceived method), 990	RouteTableReader (class in digi.xbee.models.zdo), 229
reverse() (digi.xbee.reader.RouteRecordIndicatorReceimethod), 988	ivedsi (digi.xbee.packets.raw.RX16IOPacket attribute), 366
<pre>reverse() (digi.xbee.reader.SMSReceived method),</pre>	rssi (digi.xbee.packets.raw.RX16Packet attribute), 358
982	rssi (digi.xbee.packets.raw.RX64IOPacket attribute),
reverse() (digi.xbee.reader.SocketDataReceived	361
method), 986	rssi (digi.xbee.packets.raw.RX64Packet attribute), 354
reverse() (digi.xbee.reader.SocketDataReceivedFrom method), 987	rssi (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacket attribute), 427
	run () (digi.xbee.reader.PacketListener method), 993
method), 985	running (digi.xbee.filesystem.FileProcess attribute),
reverse() (digi.xbee.reader.XBeeEvent method), 972	938
rf_data(digi.xbee.packets.common.ExplicitAddressingFattribute), 294	arkenning (digi.xbee.models.zdo.NeighborFinder at- tribute), 235
rf_data(digi.xbee.packets.common.ExplicitRXIndicatoratribute), 299	Packeting (digi.xbee.models.zdo.NeighborTableReader attribute), 233
rf_data(digi.xbee.packets.common.IODataSampleRxInd	dicatorRacket(digi.xbee.models.zdo.NodeDescriptorReader
attribute), 288	attribute), 227
<pre>rf_data (digi.xbee.packets.common.ReceivePacket at- tribute), 265</pre>	running (digi.xbee.models.zdo.RouteTableReader at- tribute), 231
rf_data(digi.xbee.packets.common.TransmitPacket at-	RX16IOPacket (class in digi.xbee.packets.raw), 363
tribute), 277	RX16Packet (class in digi.xbee.packets.raw), 356
	RX64IOPacket (class in digi.xbee.packets.raw), 360
tribute), 367	RX64Packet (class in digi.xbee.packets.raw), 353
rf_data (digi.xbee.packets.raw.RX16Packet attribute),	RXIPv4Packet (class in digi.xbee.packets.network),

336	${\tt SECURE}\ (digi.x bee. models. options. File Open Request Option$
RXSMSPacket (class in digi.xbee.packets.cellular), 247	attribute), 211
C	SECURE_SESSION_ENC
S	(digi.xbee.models.options.ReceiveOptions
<pre>scan_access_points()</pre>	attribute), 204
(digi.xbee.devices.WiFiDevice method), 749	SECURE_SESSION_ENC
scan_counter(digi.xbee.devices.AbstractXBeeDevice	(digi.xbee.models.options.RemoteATCmdOptions
attribute), 484	attribute), 207
scan_counter (digi.xbee.devices.CellularDevice at-	SECURE_SESSION_ENC (digi.xbee.models.options.TransmitOptions
tribute), 687	attribute), 206
scan_counter (digi.xbee.devices.DigiMeshDevice at-	send() (digi.xbee.sender.SyncRequestSender method),
tribute), 563	1005
scan_counter (digi.xbee.devices.DigiMeshNetwork attribute), 916	send() (digi.xbee.xsocket.socket method), 1012
scan_counter (digi.xbee.devices.DigiPointDevice at-	send_bluetooth_data()
tribute), 595	(digi.xbee.devices.CellularDevice method),
scan_counter (digi.xbee.devices.DigiPointNetwork	687
attribute), 928	send_bluetooth_data()
scan_counter (digi.xbee.devices.IPDevice attribute),	(digi.xbee.devices.DigiMeshDevice method),
661	563
scan_counter (digi.xbee.devices.LPWANDevice at-	send_bluetooth_data()
tribute), 714	(digi.xbee.devices.DigiPointDevice method),
scan_counter (digi.xbee.devices.NBIoTDevice	595
attribute), 739	send_bluetooth_data()
<pre>scan_counter (digi.xbee.devices.Raw802Device at-</pre>	(digi.xbee.devices.IPDevice method), 661
tribute), 531	send_bluetooth_data()
<pre>scan_counter (digi.xbee.devices.Raw802Network at-</pre>	(digi.xbee.devices.LPWANDevice method),
tribute), 905	714
$\verb scan_counter  (digi.xbee.devices.RemoteDigiMeshDevi$	csend_bluetooth_data()
attribute), 828	(digi.xbee.devices.NBIoTDevice method), 739
scan_counter(digi.xbee.devices.RemoteDigiPointDevic	ce 139 send_bluetooth_data()
attribute), 846	/ T
scan_counter(digi.xbee.devices.RemoteRaw802Device attribute), 810	531
scan_counter (digi.xbee.devices.RemoteXBeeDevice	send_bluetooth_data()
attribute), 792	(digi.xbee.devices.WiFiDevice method), 770
scan_counter(digi.xbee.devices.RemoteZigBeeDevice	send_bluetooth_data()
attribute), 866	(digi.xbee.devices.XBeeDevice method),
scan_counter (digi.xbee.devices.WiFiDevice at-	488
tribute), 770	send_bluetooth_data()
scan_counter (digi.xbee.devices.XBeeDevice at-	(digi.xbee.devices.ZigBeeDevice method),
tribute), 505	632
<pre>scan_counter (digi.xbee.devices.XBeeNetwork at-</pre>	send_data() (digi.xbee.devices.CellularDevice
tribute), 874	method), 688
scan_counter (digi.xbee.devices.ZigBeeDevice at-	send_data() (digi.xbee.devices.DigiMeshDevice
tribute), 632	method), 564
<pre>scan_counter (digi.xbee.devices.ZigBeeNetwork at-</pre>	send_data() (digi.xbee.devices.DigiPointDevice
tribute), 893	method), 595 send_data() (digi.xbee.devices.IPDevice method),
scan_counter_a2b (digi.xbee.devices.Connection	send_data() (digi.xbee.devices.IPDevice method), 647
attribute), 934	send_data() (digi.xbee.devices.LPWANDevice
scan_counter_b2a (digi.xbee.devices.Connection	method), 714
attribute), 934 SCAN_TIL_CANCEL (digi.xbee.devices.XBeeNetwork	send_data() (digi.xbee.devices.NBIoTDevice
attribute), 874	method), 739
···· · · · · · · · · · · · · · · · · ·	

send_	_data() <i>method</i> ), <b>5</b> 31	(digi.xbee.devices.Raw 1	802Device	(digi.xbee.devices.Raw802Device 513	method),
	771	xbee.devices.WiFiDevice		<pre>send_data_async_64_16()       (digi.xbee.devices.DigiPointDevice</pre>	method),
	487 _data()	xbee.devices.XBeeDevic (digi.xbee.devices.Zig		send_data_async_64_16() (digi.xbee.devices.ZigBeeDevice 607	method),
send_	method), 633 _data_16() method), 514	(digi.xbee.devices.Raw	802Device	send_data_broadcast() (digi.xbee.devices.CellularDevice	method),
send_	_data_64() (	(digi.xbee.devices.DigiM	<i>leshDevice</i>	688	,
send_	method), 542 _data_64() _method), 513	(digi.xbee.devices.Raw	802Device	send_data_broadcast() (digi.xbee.devices.DigiMeshDevice 565	method),
send_	_data_64_16 (digi.xbee.de 574	() evices.DigiPointDevice	method),	<pre>send_data_broadcast()      (digi.xbee.devices.DigiPointDevice      596</pre>	method),
send_	_data_64_16 (digi.xbee.de 607	() evices.ZigBeeDevice	method),	<pre>send_data_broadcast()           (digi.xbee.devices.IPDevice method), send_data_broadcast()</pre>	, 647
send_	_data_async (digi.xbee.de	() evices.CellularDevice	method),	(digi.xbee.devices.LPWANDevice 714	method),
send_	688 _data_async ( <i>digi.xbee.de</i>	() evices.DigiMeshDevice	method),	send_data_broadcast() (digi.xbee.devices.NBIoTDevice 739	method),
send_	564 _data_async (digi.xbee.de	() evices.DigiPointDevice	method),	send_data_broadcast() (digi.xbee.devices.Raw802Device 533	method),
send_	596 _data_async _method), 647	_	s.IPDevice	<pre>send_data_broadcast()           (digi.xbee.devices.WiFiDevice method send_data_broadcast()</pre>	d), 771
send_	_data_async ( <i>digi.xbee.de</i>		method),	(digi.xbee.devices.XBeeDevice 488	method),
send_	714 _data_async ( <i>digi.xbee.de</i> 739	() vices.NBIoTDevice	method),	<pre>send_data_broadcast()      (digi.xbee.devices.ZigBeeDevice      634 send_expl_data() (digi.xbee.devices.Digitations)</pre>	method),
send_	_data_async	() evices.Raw802Device	method),	method), 544 send_expl_data() (digi.xbee.devices.Digi	
send	532	() (digi.xbee.devices.V		method), 576 send_expl_data() (digi.xbee.devices.Zig.	
	method), 771	1		method), 609	
	_data_async _method), 487 _data_async		БееДечісе	send_expl_data_async() (digi.xbee.devices.DigiMeshDevice 546	method),
	_	evices.ZigBeeDevice	method),	<pre>send_expl_data_async()      (digi.xbee.devices.DigiPointDevice</pre>	method),
	_	evices.Raw802Device	method),	send_expl_data_async() (digi.xbee.devices.ZigBeeDevice 611	method),
		evices.DigiMeshDevice	method),	send_expl_data_broadcast() (digi.xbee.devices.DigiMeshDevice 545	method),
~ ~ · · · · · · _	_~~~~~~~~~~~	_~ · · \ /		0.10	

send_expl_data_broadcast()	597
(digi.xbee.devices.DigiPointDevice method),	send_micropython_data()
577	(digi.xbee.devices.IPDevice method), 662
<pre>send_expl_data_broadcast()     (digi.xbee.devices.ZigBeeDevice method),</pre>	<pre>send_micropython_data()      (digi.xbee.devices.LPWANDevice method),</pre>
610	715
send_file_xmodem() (in module	send_micropython_data()
digi.xbee.util.xmodem), 462	(digi.xbee.devices.NBIoTDevice method),
send_file_ymodem() (in module	740
digi.xbee.util.xmodem), 462	send_micropython_data()
<pre>send_ip_data() (digi.xbee.devices.CellularDevice</pre>	(digi.xbee.devices.Raw802Device method), 533
send_ip_data() (digi.xbee.devices.IPDevice	send_micropython_data()
method), 644	(digi.xbee.devices.WiFiDevice method), 772
send_ip_data() (digi.xbee.devices.LPWANDevice	send_micropython_data()
method), 696	(digi.xbee.devices.XBeeDevice method),
send_ip_data() (digi.xbee.devices.NBIoTDevice	489
method), 739	<pre>send_micropython_data()     (digi.xbee.devices.ZigBeeDevice method),</pre>
send_ip_data() (digi.xbee.devices.WiFiDevice method),771	(digi.xbee.devices.ZigBeeDevice method), 634
send_ip_data_async()	send_multicast_data()
(digi.xbee.devices.CellularDevice method),	(digi.xbee.devices.ZigBeeDevice method),
688	611
<pre>send_ip_data_async()</pre>	send_multicast_data_async()
(digi.xbee.devices.IPDevice method), 645	(digi.xbee.devices.ZigBeeDevice method),
<pre>send_ip_data_async()</pre>	612
(digi.xbee.devices.LPWANDevice method), 697	<pre>send_packet() (digi.xbee.devices.CellularDevice</pre>
send_ip_data_async()	<pre>send_packet() (digi.xbee.devices.DigiMeshDevice</pre>
(digi.xbee.devices.NBIoTDevice method),	method), 565
740	send_packet() (digi.xbee.devices.DigiPointDevice
send_ip_data_async()	method), 597
<pre>(digi.xbee.devices.WiFiDevice method), 771 send_ip_data_broadcast()</pre>	send_packet() (digi.xbee.devices.IPDevice method), 662
(digi.xbee.devices.CellularDevice method),	send_packet() (digi.xbee.devices.LPWANDevice
689	method), 715
send_ip_data_broadcast()	send_packet() (digi.xbee.devices.NBIoTDevice
(digi.xbee.devices.IPDevice method), 645	method), 741
send_ip_data_broadcast()	send_packet() (digi.xbee.devices.Raw802Device
(digi.xbee.devices.LPWANDevice method),	method), 533
714	send_packet() (digi.xbee.devices.WiFiDevice
<pre>send_ip_data_broadcast()     (digi.xbee.devices.NBIoTDevice method),</pre>	method), 772 send_packet() (digi.xbee.devices.XBeeDevice
740	method), 494
send_ip_data_broadcast()	send_packet() (digi.xbee.devices.ZigBeeDevice
(digi.xbee.devices.WiFiDevice method), 772	method), 634
send_micropython_data()	send_packet() (digi.xbee.sender.PacketSender
(digi.xbee.devices.CellularDevice method),	method), 1004
689	<pre>send_packet_sync_and_get_response()</pre>
send_micropython_data()	$(digi.x bee. devices. Cellular Device\ method), 690$
(digi.xbee.devices.DigiMeshDevice method), 565	<pre>send_packet_sync_and_get_response()      (digi.xbee.devices.DigiMeshDevice method),</pre>
send_micropython_data()	566
(digi.xbee.devices.DigiPointDevice method),	<pre>send_packet_sync_and_get_response()</pre>

(digi.xbee.devices.DigiPointDevice method), 598	(digi.xbee.devices.ZigBeeDevice method),
send_packet_sync_and_get_response()	sendall() (digi.xbee.xsocket.socket method), 1012
(digi.xbee.devices.IPDevice method), 663	SendDataRequestOptions (class in
send_packet_sync_and_get_response()	digi.xbee.models.options), 207
(digi.xbee.devices.LPWANDevice method), 716	SendDataRequestPacket (class in
· ·	
send_packet_sync_and_get_response()	digi.xbee.packets.devicecloud), 312
(digi.xbee.devices.NBIoTDevice method), 741	SendDataResponsePacket (class in
send_packet_sync_and_get_response()	digi.xbee.packets.devicecloud), 315
(digi.xbee.devices.Raw802Device method), 534	sendto() (digi.xbee.xsocket.socket method), 1012
<pre>send_packet_sync_and_get_response()</pre>	SENT_FROM_END_DEVICE
(digi.xbee.devices.WiFiDevice method), 773	(digi.xbee.models.options.ReceiveOptions
<pre>send_packet_sync_and_get_response()</pre>	attribute), 204
(digi.xbee.devices.XBeeDevice method), 494	serial_port (digi.xbee.devices.CellularDevice at-
<pre>send_packet_sync_and_get_response()</pre>	tribute), 691
(digi.xbee.devices.ZigBeeDevice method), 635	serial_port (digi.xbee.devices.DigiMeshDevice at-
send_sms() (digi.xbee.devices.CellularDevice	tribute), 567
method), 671	serial_port (digi.xbee.devices.DigiPointDevice at-
send_sms() (digi.xbee.devices.LPWANDevice	tribute), 598
method), 697	serial_port (digi.xbee.devices.IPDevice attribute),
send_sms() (digi.xbee.devices.NBIoTDevice method),	663
742	serial_port (digi.xbee.devices.LPWANDevice
<pre>send_sms_async() (digi.xbee.devices.CellularDevice</pre>	attribute), 716
method), 671	serial_port (digi.xbee.devices.NBIoTDevice at-
<pre>send_sms_async() (digi.xbee.devices.LPWANDevice</pre>	tribute), 742
method), 697	serial_port (digi.xbee.devices.Raw802Device
<pre>send_sms_async() (digi.xbee.devices.NBIoTDevice</pre>	attribute), 535
method), 742	serial_port (digi.xbee.devices.WiFiDevice at-
<pre>send_user_data_relay()</pre>	tribute), 774
(digi.xbee.devices.CellularDevice method),	serial_port (digi.xbee.devices.XBeeDevice at-
690	tribute), 486
<pre>send_user_data_relay()</pre>	serial_port (digi.xbee.devices.ZigBeeDevice at-
(digi.xbee.devices.DigiMeshDevice method),	tribute), 636
566	set_16bit_addr() (digi.xbee.devices.AbstractXBeeDevice
<pre>send_user_data_relay()</pre>	method), 472
(digi.xbee.devices.DigiPointDevice method),	set_16bit_addr() (digi.xbee.devices.CellularDevice
598	method), 691
send_user_data_relay()	<pre>set_16bit_addr() (digi.xbee.devices.DigiMeshDevice</pre>
(digi.xbee.devices.IPDevice method), 663	method), 567
send_user_data_relay()	<pre>set_16bit_addr() (digi.xbee.devices.DigiPointDevice</pre>
(digi.xbee.devices.LPWANDevice method),	method), 599
716	set_16bit_addr() (digi.xbee.devices.IPDevice
send_user_data_relay()	method), 664
(digi.xbee.devices.NBIoTDevice method),	set_16bit_addr() (digi.xbee.devices.LPWANDevice
742	method), 717
send_user_data_relay()	set_16bit_addr() (digi.xbee.devices.NBIoTDevice
(digi.xbee.devices.Raw802Device method),	method), 742
(digi.xbee.devices.Rawoo2Device memod), 534	set_16bit_addr() (digi.xbee.devices.Raw802Device
send_user_data_relay()  (digitable devices WiFiDevice method) 773	method), 535
(digi.xbee.devices.WiFiDevice method), 773	set_16bit_addr() (digi.xbee.devices.RemoteDigiMeshDevice
send_user_data_relay()  (digitable devices VPasDevice method)	method), 828
(digi.xbee.devices.XBeeDevice method),	set_16bit_addr() (digi.xbee.devices.RemoteDigiPointDevice
488	method), 846
send user data relay()	set 16bit addr() (digi.xbee.devices.RemoteRaw802Device

```
method), 810
                                                           (digi.xbee.devices.WiFiDevice method), 774
set 16bit addr() (digi.xbee.devices.RemoteXBeeDevixet api output mode()
                                                           (digi.xbee.devices.XBeeDevice
        method), 792
                                                                                           method),
set_16bit_addr() (digi.xbee.devices.RemoteZigBeeDevice
                                                           506
        method), 866
                                                  set_api_output_mode()
set 16bit addr()
                       (digi.xbee.devices.WiFiDevice
                                                           (digi.xbee.devices.ZigBeeDevice
                                                                                           method),
        method), 774
set_16bit_addr()
                                                  set_api_output_mode_value()
                      (digi.xbee.devices.XBeeDevice
        method), 506
                                                           (digi.xbee.devices.AbstractXBeeDevice
set_16bit_addr() (digi.xbee.devices.ZigBeeDevice
                                                           method), 481
        method), 636
                                                  set_api_output_mode_value()
set_64bit_addr() (digi.xbee.devices.RemoteRaw802Device
                                                           (digi.xbee.devices.CellularDevice
                                                                                           method),
        method), 799
set_access_point_timeout()
                                                  set_api_output_mode_value()
        (digi.xbee.devices.WiFiDevice method), 752
                                                           (digi.xbee.devices.DigiMeshDevice
                                                                                           method),
set_api_output_mode()
                                                           568
        (digi.xbee.devices.AbstractXBeeDevice
                                                  set_api_output_mode_value()
        method), 481
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                           method),
set_api_output_mode()
                                                  set_api_output_mode_value()
        (digi.xbee.devices.CellularDevice
                                        method),
        691
                                                           (digi.xbee.devices.IPDevice method), 664
set_api_output_mode()
                                                  set_api_output_mode_value()
                                                           (digi.xbee.devices.LPWANDevice
        (digi.xbee.devices.DigiMeshDevice
                                        method),
                                                                                           method),
set_api_output_mode()
                                                  set_api_output_mode_value()
                                                           (digi.xbee.devices.NBIoTDevice
        (digi.xbee.devices.DigiPointDevice
                                        method),
                                                                                           method),
set_api_output_mode()
                                                  set_api_output_mode_value()
                                                           (digi.xbee.devices.Raw802Device
        (digi.xbee.devices.IPDevice method), 664
                                                                                           method),
set_api_output_mode()
                                                           536
        (digi.xbee.devices.LPWANDevice
                                        method),
                                                  set_api_output_mode_value()
        717
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
set_api_output_mode()
                                                           method), 829
        (digi.xbee.devices.NBIoTDevice
                                        method),
                                                  set_api_output_mode_value()
        743
                                                           (digi.xbee.devices.RemoteDigiPointDevice
                                                           method), 847
set_api_output_mode()
        (digi.xbee.devices.Raw802Device
                                        method),
                                                  set_api_output_mode_value()
        535
                                                           (digi.xbee.devices.RemoteRaw802Device
set_api_output_mode()
                                                           method), 810
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                  set_api_output_mode_value()
        method), 829
                                                           (digi.xbee.devices.RemoteXBeeDevice
set_api_output_mode()
                                                           method), 793
        (digi.xbee.devices.RemoteDigiPointDevice
                                                  set_api_output_mode_value()
        method), 847
                                                           (digi.xbee.devices.RemoteZigBeeDevice
set_api_output_mode()
                                                           method), 867
        (digi.xbee.devices.RemoteRaw802Device
                                                  set_api_output_mode_value()
                                                           (digi.xbee.devices.WiFiDevice method), 775
        method), 810
set_api_output_mode()
                                                  set_api_output_mode_value()
        (digi.xbee.devices.RemoteXBeeDevice
                                                           (digi.xbee.devices.XBeeDevice
                                                                                           method),
        method), 792
set_api_output_mode()
                                                  set_api_output_mode_value()
        (digi.xbee.devices.RemoteZigBeeDevice
                                                           (digi.xbee.devices.ZigBeeDevice
                                                                                           method),
        method), 867
                                                           637
set_api_output_mode()
                                                  set baudrate()
                                                                       (digi.xbee.serial.XBeeSerialPort
```

method), 1007		set_dest_address()	
set_deep_discovery_options()		(digi.xbee.devices.RemoteDigiMeshDevic	ce
(digi.xbee.devices.DigiMeshNetwork	method),	method), 830	
916		<pre>set_dest_address()</pre>	
<pre>set_deep_discovery_options()</pre>		(digi.xbee.devices.RemoteDigiPointDevic	ce :
(digi.xbee.devices.DigiPointNetwork	method),	method), 848	
928		<pre>set_dest_address()</pre>	
<pre>set_deep_discovery_options()</pre>		(digi.xbee.devices.RemoteRaw802Device	
(digi.xbee.devices.Raw802Network	method),	method), 811	
905		set_dest_address()	
<pre>set_deep_discovery_options()</pre>		(digi.xbee.devices.RemoteXBeeDevice	
	method),	method), 793	
880		set_dest_address()	
<pre>set_deep_discovery_options()</pre>		(digi.xbee.devices.RemoteZigBeeDevice	
	method),	method), 867	
894		<pre>set_dest_address()</pre>	
<pre>set_deep_discovery_timeouts()</pre>		(digi.xbee.devices.WiFiDevice method), 7	775
(digi.xbee.devices.DigiMeshNetwork	method),	<pre>set_dest_address()</pre>	.1 1
917		· · · · · · · · · · · · · · · · · · ·	ethod),
<pre>set_deep_discovery_timeouts()</pre>	.1 1	507	
(digi.xbee.devices.DigiPointNetwork	method),	set_dest_address()	.1 N
928			ethod),
set_deep_discovery_timeouts()	41 41	637	
(digi.xbee.devices.Raw802Network	method),	set_dest_ip_addr()	. 4la o d
		(digi.xbee.devices.CellularDevice me 692	ethod),
set_deep_discovery_timeouts()	mathad)		Daviaa
(digi.xbee.devices.XBeeNetwork 882	method),	<pre>set_dest_ip_addr() (digi.xbee.devices.IPI</pre>	Device
set_deep_discovery_timeouts()		set_dest_ip_addr()	
	method),		ethod),
894	memou),	718	mou),
set_dest_address()		set_dest_ip_addr()	
(digi.xbee.devices.AbstractXBeeDevice	2		ethod),
method), 473		744	,,
set_dest_address()		set_dest_ip_addr()	
	method),	(digi.xbee.devices.WiFiDevice method), 7	775
692	,	set_dio_change_detection()	
<pre>set_dest_address()</pre>		(digi.xbee.devices.AbstractXBeeDevice	
(digi.xbee.devices.DigiMeshDevice	method),	method), 480	
568		<pre>set_dio_change_detection()</pre>	
set_dest_address()		(digi.xbee.devices.CellularDevice me	ethod),
(digi.xbee.devices.DigiPointDevice	method),	672	
600		<pre>set_dio_change_detection()</pre>	
set_dest_address() (digi.xbee.devices.	IPDevice		ethod),
method), 646		569	
<pre>set_dest_address()</pre>		<pre>set_dio_change_detection()</pre>	
	method),		ethod),
718		600	
<pre>set_dest_address()</pre>	,	<pre>set_dio_change_detection()</pre>	_
	method),	(digi.xbee.devices.IPDevice method), 665	)
744		set_dio_change_detection()	-41 N
set_dest_address()	math a A		ethod),
	method),	718	
536		<pre>set_dio_change_detection()</pre>	

```
(digi.xbee.devices.NBIoTDevice
                                         method),
                                                  set_dio_value()
                                                                          (digi.xbee.devices.WiFiDevice
        744
                                                           method), 776
set dio change detection()
                                                   set_dio_value()
                                                                         (digi.xbee.devices.XBeeDevice
        (digi.xbee.devices.Raw802Device
                                                           method), 507
                                         method),
                                                   set_dio_value()
                                                                        (digi.xbee.devices.ZigBeeDevice
set dio change detection()
                                                           method), 638
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                   set_discovery_options()
        method), 830
                                                           (digi.xbee.devices.DigiMeshNetwork method),
set_dio_change_detection()
                                                           917
        (digi.xbee.devices.RemoteDigiPointDevice
                                                   set_discovery_options()
        method), 848
                                                           (digi.xbee.devices.DigiPointNetwork method),
                                                           929
set_dio_change_detection()
        (digi.xbee.devices.RemoteRaw802Device
                                                   set_discovery_options()
        method), 811
                                                           (digi.xbee.devices.Raw802Network
                                                                                            method),
set_dio_change_detection()
        (digi.xbee.devices.RemoteXBeeDevice
                                                   set_discovery_options()
        method), 794
                                                           (digi.xbee.devices.XBeeNetwork
                                                                                            method),
                                                           880
set_dio_change_detection()
        (digi.xbee.devices.RemoteZigBeeDevice
                                                   set_discovery_options()
        method), 868
                                                           (digi.xbee.devices.ZigBeeNetwork
                                                                                            method),
set_dio_change_detection()
                                                           895
        (digi.xbee.devices.WiFiDevice method), 776
                                                   set_discovery_timeout()
set_dio_change_detection()
                                                           (digi.xbee.devices.DigiMeshNetwork method),
        (digi.xbee.devices.XBeeDevice
                                         method).
        507
                                                   set_discovery_timeout()
set_dio_change_detection()
                                                           (digi.xbee.devices.DigiPointNetwork method),
        (digi.xbee.devices.ZigBeeDevice
                                         method),
                                                   set_discovery_timeout()
set_dio_value() (digi.xbee.devices.AbstractXBeeDevice
                                                           (digi.xbee.devices.Raw802Network
                                                                                            method),
        method), 479
                                                           906
set_dio_value() (digi.xbee.devices.CellularDevice
                                                  set_discovery_timeout()
        method), 692
                                                           (digi.xbee.devices.XBeeNetwork
                                                                                            method),
set_dio_value() (digi.xbee.devices.DigiMeshDevice
                                                           881
        method), 569
                                                   set_discovery_timeout()
set_dio_value() (digi.xbee.devices.DigiPointDevice
                                                           (digi.xbee.devices.ZigBeeNetwork
                                                                                            method),
        method), 601
set_dio_value()
                         (digi.xbee.devices.IPDevice
                                                   set_dns_address() (digi.xbee.devices.WiFiDevice
        method), 665
                                                           method), 754
set_dio_value() (digi.xbee.devices.LPWANDevice
                                                  set_gateway_address()
                                                           (digi.xbee.devices.WiFiDevice method), 754
        method), 718
set_dio_value()
                     (digi.xbee.devices.NBIoTDevice
                                                  set io configuration()
        method), 744
                                                           (digi.xbee.devices.AbstractXBeeDevice
                                                           method), 475
set_dio_value() (digi.xbee.devices.Raw802Device
                                                   set_io_configuration()
        method), 537
                                                           (digi.xbee.devices.CellularDevice
set_dio_value() (digi.xbee.devices.RemoteDigiMeshDevice
                                                                                            method),
                                                           693
        method), 830
set_dio_value() (digi.xbee.devices.RemoteDigiPointDevice_io_configuration()
                                                           (digi.xbee.devices.DigiMeshDevice
        method), 848
                                                                                            method),
set_dio_value() (digi.xbee.devices.RemoteRaw802Device
                                                           569
        method), 812
                                                   set_io_configuration()
set_dio_value() (digi.xbee.devices.RemoteXBeeDevice
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                            method),
        method), 794
set_dio_value() (digi.xbee.devices.RemoteZigBeeDevicet_io_configuration()
        method), 868
                                                           (digi.xbee.devices.IPDevice method), 665
```

set_io_	_configuration() (digi.xbee.devices.LPWANDevice 719	method),	538 set_io_sampling_rate() (digi.xbee.devices.RemoteDigiMeshDevice
set_io	_configuration() (digi.xbee.devices.NBIoTDevice 744	method),	method), 831 set_io_sampling_rate() (digi.xbee.devices.RemoteDigiPointDevice
set_io_	_configuration()		method), 849
	(digi.xbee.devices.Raw802Device 537	method),	set_io_sampling_rate() (digi.xbee.devices.RemoteRaw802Device
	_configuration() (digi.xbee.devices.RemoteDigiMeshD method), 831	evice	<pre>method), 812 set_io_sampling_rate()    (digi.xbee.devices.RemoteXBeeDevice</pre>
	_configuration() (digi.xbee.devices.RemoteDigiPointDomethod), 849	evice	<pre>method), 795 set_io_sampling_rate()     (digi.xbee.devices.RemoteZigBeeDevice)</pre>
set_io	_configuration() (digi.xbee.devices.RemoteRaw802Dev method), 812	rice	<pre>method), 869 set_io_sampling_rate()   (digi.xbee.devices.WiFiDevice method), 777</pre>
set_io	_configuration() (digi.xbee.devices.RemoteXBeeDevice method), 795	2	<pre>set_io_sampling_rate()     (digi.xbee.devices.XBeeDevice method),     508</pre>
set_io	_configuration() (digi.xbee.devices.RemoteZigBeeDevi method), 869	ce	<pre>set_io_sampling_rate()     (digi.xbee.devices.ZigBeeDevice method),     639</pre>
set_io_	_configuration() (digi.xbee.devices.WiFiDevice method	l), 776	<pre>set_ip_address() (digi.xbee.devices.WiFiDevice</pre>
set_io	_configuration() (digi.xbee.devices.XBeeDevice 508	method),	<pre>set_ip_addressing_mode()           (digi.xbee.devices.WiFiDevice method), 752 set_local_xbee_device()</pre>
set_io_	_configuration() (digi.xbee.devices.ZigBeeDevice 639	method),	(digi.xbee.devices.RemoteDigiMeshDevice method), 832 set_local_xbee_device()
set_io	_sampling_rate() (digi.xbee.devices.AbstractXBeeDevice method), 477	re	(digi.xbee.devices.RemoteDigiPointDevice method), 850 set_local_xbee_device()
set_io_	_sampling_rate() (digi.xbee.devices.CellularDevice 672	method),	(digi.xbee.devices.RemoteRaw802Device method), 813 set_local_xbee_device()
set_io	_sampling_rate() (digi.xbee.devices.DigiMeshDevice 570	method),	(digi.xbee.devices.RemoteXBeeDevice method), 781 set_local_xbee_device()
set_io_	_sampling_rate() (digi.xbee.devices.DigiPointDevice 601	method),	(digi.xbee.devices.RemoteZigBeeDevice method), 869 set_many_to_one_broadcasting_time()
set_io_	_sampling_rate() (digi.xbee.devices.IPDevice method),	666	(digi.xbee.devices.ZigBeeDevice method), 607 set_mask_address()
set_io	_sampling_rate() (digi.xbee.devices.LPWANDevice 719	method),	(digi.xbee.devices.WiFiDevice method), 753 set_node_id() (digi.xbee.devices.AbstractXBeeDevice method), 471
set_io	_sampling_rate() (digi.xbee.devices.NBIoTDevice 745	method),	<pre>set_node_id() (digi.xbee.devices.CellularDevice</pre>
set_io	_sampling_rate() (digi.xbee.devices.Raw802Device	method),	method), 570 set_node_id() (digi.xbee.devices.DigiPointDevice

method), 602	set_pan_id() (digi.xbee.devices.Raw802Device
<pre>set_node_id() (digi.xbee.devices.IPDevice method), 666</pre>	<pre>method), 538 set_pan_id() (digi.xbee.devices.RemoteDigiMeshDevice</pre>
set_node_id() (digi.xbee.devices.LPWANDevice	method), 832
<pre>method), 719 set_node_id()</pre>	<pre>set_pan_id() (digi.xbee.devices.RemoteDigiPointDevice</pre>
method), 745	set_pan_id() (digi.xbee.devices.RemoteRaw802Device
<pre>set_node_id() (digi.xbee.devices.Raw802Device     method), 538</pre>	<pre>method), 813 set_pan_id() (digi.xbee.devices.RemoteXBeeDevice</pre>
set_node_id() (digi.xbee.devices.RemoteDigiMeshDe	vice method), 796
<pre>method), 832 set_node_id() (digi.xbee.devices.RemoteDigiPointDe</pre>	<pre>set_pan_id() (digi.xbee.devices.RemoteZigBeeDevice vice</pre>
method), 850	set_pan_id() (digi.xbee.devices.WiFiDevice
<pre>set_node_id() (digi.xbee.devices.RemoteRaw802Devi method), 813</pre>	ce method), 777 set_pan_id() (digi.xbee.devices.XBeeDevice
<pre>set_node_id() (digi.xbee.devices.RemoteXBeeDevice</pre>	method), 509
<pre>method), 795 set_node_id() (digi.xbee.devices.RemoteZigBeeDevic</pre>	set_pan_id() (digi.xbee.devices.ZigBeeDevice e method), 640
method), 870	set_parameter()(digi.xbee.devices.AbstractXBeeDevice
set_node_id() (digi.xbee.devices.WiFiDevice method),777	method), 468 set_parameter() (digi.xbee.devices.CellularDevice
set_node_id() (digi.xbee.devices.XBeeDevice	method), 693
<pre>method), 509 set_node_id() (digi.xbee.devices.ZigBeeDevice</pre>	<pre>set_parameter() (digi.xbee.devices.DigiMeshDevice     method), 571</pre>
method), 639	set_parameter() (digi.xbee.devices.DigiPointDevice
<pre>set_ota_max_block_size()   (digi.xbee.devices.RemoteDigiMeshDevice</pre>	method), 602 set_parameter() (digi.xbee.devices.IPDevice
method), 832	method), 667
<pre>set_ota_max_block_size()      (digi.xbee.devices.RemoteDigiPointDevice</pre>	<pre>set_parameter() (digi.xbee.devices.LPWANDevice     method), 719</pre>
method), 850	set_parameter() (digi.xbee.devices.NBIoTDevice
<pre>set_ota_max_block_size()     (digi.xbee.devices.RemoteRaw802Device</pre>	<pre>method), 745 set_parameter() (digi.xbee.devices.Raw802Device</pre>
method), 813	method), 539
<pre>set_ota_max_block_size()     (digi.xbee.devices.RemoteXBeeDevice</pre>	<pre>set_parameter() (digi.xbee.devices.RemoteDigiMeshDevice</pre>
method), 782	<pre>set_parameter() (digi.xbee.devices.RemoteDigiPointDevice</pre>
<pre>set_ota_max_block_size()     (digi.xbee.devices.RemoteZigBeeDevice</pre>	<pre>method), 850 set_parameter() (digi.xbee.devices.RemoteRaw802Device</pre>
method), 870	method), 814
<pre>set_pan_id() (digi.xbee.devices.AbstractXBeeDevice     method), 474</pre>	<pre>set_parameter() (digi.xbee.devices.RemoteXBeeDevice</pre>
set_pan_id() (digi.xbee.devices.CellularDevice	set_parameter() (digi.xbee.devices.RemoteZigBeeDevice
<pre>method), 693 set_pan_id() (digi.xbee.devices.DigiMeshDevice</pre>	method), 870 set_parameter() (digi.xbee.devices.WiFiDevice
method), 570	method), 778
<pre>set_pan_id() (digi.xbee.devices.DigiPointDevice     method), 602</pre>	set_parameter() (digi.xbee.devices.XBeeDevice method), 486
<pre>set_pan_id() (digi.xbee.devices.IPDevice method),</pre>	set_parameter() (digi.xbee.devices.ZigBeeDevice
647 set_pan_id() (digi.xbee.devices.LPWANDevice	<pre>method), 640 set_power_level()</pre>
method), 719	(digi.xbee.devices. Abstract XBee Device
set_pan_id() (digi.xbee.devices.NBIoTDevice method), 745	<pre>method), 475 set_power_level()</pre>

(digi.xbee.devices.CellularDevice ma 673	ethod),	(digi.xbee.devices.IPDevice method), 667 set_pwm_duty_cycle()
<pre>set_power_level()      (digi.xbee.devices.DigiMeshDevice ma 571</pre>	ethod),	(digi.xbee.devices.LPWANDevice method), 719 set_pwm_duty_cycle()
<pre>set_power_level()      (digi.xbee.devices.DigiPointDevice ma 603</pre>	ethod),	(digi.xbee.devices.NBIoTDevice method), 745 set_pwm_duty_cycle()
<pre>set_power_level() (digi.xbee.devices.IP.</pre>	Device	(digi.xbee.devices.Raw802Device method), 539
<pre>set_power_level()     (digi.xbee.devices.LPWANDevice</pre>	ethod),	<pre>set_pwm_duty_cycle()      (digi.xbee.devices.RemoteDigiMeshDevice      method), 833</pre>
<pre>set_power_level()     (digi.xbee.devices.NBIoTDevice</pre>	ethod),	<pre>set_pwm_duty_cycle()     (digi.xbee.devices.RemoteDigiPointDevice     method), 851</pre>
<pre>set_power_level()     (digi.xbee.devices.Raw802Device</pre>	ethod),	<pre>set_pwm_duty_cycle()     (digi.xbee.devices.RemoteRaw802Device     method), 814</pre>
<pre>set_power_level()     (digi.xbee.devices.RemoteDigiMeshDevices.RemoteDigimeshDevices.RemoteDigimeshDevices.RemoteDigimeshDevices.RemoteDigimeshDevices.RemoteDigimeshDevices.RemoteDigimeshDevices.RemoteDigimeshDevices.RemoteDigimeshDevice</pre>	ce	<pre>set_pwm_duty_cycle()     (digi.xbee.devices.RemoteXBeeDevice     method), 796</pre>
<pre>set_power_level()     (digi.xbee.devices.RemoteDigiPointDevice     method), 851</pre>	ce	<pre>set_pwm_duty_cycle()      (digi.xbee.devices.RemoteZigBeeDevice      method), 871</pre>
<pre>set_power_level()      (digi.xbee.devices.RemoteRaw802Device      method), 814</pre>	2	<pre>set_pwm_duty_cycle()           (digi.xbee.devices.WiFiDevice method), 778 set_pwm_duty_cycle()</pre>
<pre>set_power_level()      (digi.xbee.devices.RemoteXBeeDevice      method), 796</pre>		(digi.xbee.devices.XBeeDevice method), 510 set_pwm_duty_cycle()
<pre>set_power_level()      (digi.xbee.devices.RemoteZigBeeDevice</pre>		(digi.xbee.devices.ZigBeeDevice method), 640
<pre>method), 871 set_power_level() (digi.xbee.devices.WiFi.     method), 778</pre>	Device	<pre>set_read_timeout()     (digi.xbee.serial.XBeeSerialPort method),</pre>
<pre>set_power_level() (digi.xbee.devices.XBee.</pre>	Device	<pre>set_sync_ops_timeout()      (digi.xbee.devices.AbstractXBeeDevice      method), 473</pre>
(digi.xbee.devices.ZigBeeDevice ma 640	ethod),	<pre>set_sync_ops_timeout()      (digi.xbee.devices.CellularDevice method),</pre>
<pre>set_pwm_duty_cycle()     (digi.xbee.devices.AbstractXBeeDevice     method), 478</pre>		694 set_sync_ops_timeout() (digi.xbee.devices.DigiMeshDevice method),
693	ethod),	572 set_sync_ops_timeout() (digi.xbee.devices.DigiPointDevice method),
<pre>set_pwm_duty_cycle()     (digi.xbee.devices.DigiMeshDevice mage) 571</pre>	ethod),	603 set_sync_ops_timeout() (digi.xbee.devices.IPDevice method), 667
set_pwm_duty_cycle()	ethod),	set_sync_ops_timeout() (digi.xbee.devices.LPWANDevice method), 720
set_pwm_duty_cycle()		set_sync_ops_timeout()

(digi.xbee.devices.NBIoTDevice method), 746	<pre>attribute), 398 socket_id (digi.xbee.packets.socket.SocketConnectPacket</pre>
set_sync_ops_timeout()	attribute), 389
(digi.xbee.devices.Raw802Device method),	$\verb socket_id  \textit{(digi.xbee.packets.socket.SocketConnectResponsePacket }$
540	attribute), 392
<pre>set_sync_ops_timeout()</pre>	<pre>socket_id(digi.xbee.packets.socket.SocketCreateResponsePacket</pre>
(digi.x bee. devices. Remote DigiMesh Device	attribute), 378
method), 833	<pre>socket_id(digi.xbee.packets.socket.SocketListenResponsePacket</pre>
<pre>set_sync_ops_timeout()</pre>	attribute), 411
(digi.xbee.devices.RemoteDigiPointDevice	<pre>socket_id(digi.xbee.packets.socket.SocketNewIPv4ClientPacket</pre>
method), 851	attribute), 414
<pre>set_sync_ops_timeout()</pre>	<pre>socket_id(digi.xbee.packets.socket.SocketOptionRequestPacket</pre>
(digi.xbee.devices.RemoteRaw802Device	attribute), 381
method), 815	<pre>socket_id(digi.xbee.packets.socket.SocketOptionResponsePacket</pre>
set_sync_ops_timeout()	attribute), 385
(digi.xbee.devices.RemoteXBeeDevice	socket_id(digi.xbee.packets.socket.SocketReceiveFromPacket
method), 797	attribute), 422
set_sync_ops_timeout()	socket_id(digi.xbee.packets.socket.SocketReceivePacket
(digi.xbee.devices.RemoteZigBeeDevice	attribute), 419
method), 871	socket_id (digi.xbee.packets.socket.SocketSendPacket
set_sync_ops_timeout()	attribute), 401
(digi.xbee.devices.WiFiDevice method), 778	socket_id(digi.xbee.packets.socket.SocketSendToPacket
set_sync_ops_timeout()	attribute), 404
(digi.xbee.devices.XBeeDevice method),	socket_id (digi.xbee.packets.socket.SocketStatePacket
510	attribute), 425
set_sync_ops_timeout()	SocketBindListenPacket (class in
(digi.xbee.devices.ZigBeeDevice method),	digi.xbee.packets.socket), 406
641	SocketClosePacket (class in
setblocking() (digi.xbee.xsocket.socket method),	digi.xbee.packets.socket), 394
1011	SocketCloseResponsePacket (class in
setsocketopt() (digi.xbee.xsocket.socket method),	digi.xbee.packets.socket), 397
1013	SocketConnectPacket (class in
settimeout() (digi.xbee.xsocket.socket method),	digi.xbee.packets.socket), 387
1011	SocketConnectResponsePacket (class in
signal_quality(digi.xbee.models.accesspoint.Access.	
attribute), 125	SocketCreatePacket (class in
size (digi.xbee.filesystem.FileSystemElement attribute),	digi.xbee.packets.socket), 374
937	SocketCreateResponsePacket (class in
size (digi.xbee.models.filesystem.OpenFileCmdResponse	digi.xbee.packets.socket), 376
attribute), 140	SocketDataReceived (class in digi.xbee.reader),
size (digi.xbee.models.filesystem.ReadFileCmdRequest	985
attribute), 145	SocketDataReceivedFrom (class in
size_pretty (digi.xbee.filesystem.FileSystemElement	digi.xbee.reader), 986
attribute), 937	SocketInfo (class in digi.xbee.models.info), 190
SMSMessage (class in digi.xbee.models.message), 202	SocketInfo(class in digi.xbee.models.status),
	225
SMSReceived (class in digi.xbee.reader), 981 socket (class in digi.xbee.xsocket), 1010	SocketListenResponsePacket (class in
	digi.xbee.packets.socket), 409
tribute), 191	·
socket_id(digi.xbee.packets.socket.SocketBindListenPa	
attribute), 408	SocketOption (class in digi.xbee.models.options), 209
socket_id(digi.xbee.packets.socket.SocketClosePacket	
attribute), 395	SocketOptionRequestPacket (class in

SocketOptionResponsePacket (class in	method), 987
digi.xbee.packets.socket), 383	sort () (digi.xbee.reader.SocketStateReceived method),
SocketReceiveFromPacket (class in	985
digi.xbee.packets.socket), 419 SocketReceivePacket (class in	sort () (digi.xbee.reader.XBeeEvent method), 972 source_address (digi.xbee.packets.network.RXIPv4Packet
digi.xbee.packets.socket), 416	attribute), 337
	source_address (digi.xbee.packets.socket.SocketReceiveFromPacket
digi.xbee.packets.socket), 400	attribute), 422
SocketSendToPacket (class in	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifi.
digi.xbee.packets.socket), 403	attribute), 427
SocketState (class in digi.xbee.models.status), 225	source_address(digi.xbee.packets.wifi.RemoteATCommandResponseV
SocketStatePacket (class in	attribute), 436
digi.xbee.packets.socket), 422	source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage
SocketStateReceived (class in digi.xbee.reader),	attribute), 200
984	source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacket
SocketStatus (class in digi.xbee.models.status), 224	attribute), 294
sort() (digi.xbee.reader.BluetoothDataReceived	source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPack
method), 983 sort () (digi.xbee.reader.DataReceived method), 975	attribute), 298 source_port (digi.xbee.models.message.IPMessage
sort () (digi.xbee.reader.DeviceDiscovered method),	attribute), 201
978	source_port (digi.xbee.packets.network.RXIPv4Packet
sort() (digi.xbee.reader.DiscoveryProcessFinished	attribute), 338
method), 979	source_port (digi.xbee.packets.network.TXIPv4Packet
<pre>sort() (digi.xbee.reader.EndDiscoveryScan method),</pre>	attribute), 343
991	<pre>source_port (digi.xbee.packets.socket.SocketBindListenPacket</pre>
sort() (digi.xbee.reader.ExplicitDataReceived	attribute), 408
method), 980	source_port (digi.xbee.packets.socket.SocketReceiveFromPacket
sort() (digi.xbee.reader.FileSystemFrameReceived	attribute), 422
method), 992	SPECIAL_FUNCTIONALITY (digi.xbee.io.IOMode at-
sort() (digi.xbee.reader.InitDiscoveryScan method),	tribute), 962
991 sort() (digi.xbee.reader.IOSampleReceived method),	SpecialByte (class in digi.xbee.models.atcomm), 129 src_addr (digi.xbee.packets.digimesh.RouteInformationPacket
977	attribute), 322
sort () (digi.xbee.reader.IPDataReceived method), 981	src_event (digi.xbee.packets.digimesh.RouteInformationPacket
sort () (digi.xbee.reader.MicroPythonDataReceived	attribute), 320
method), 984	<pre>src_interface(digi.xbee.packets.relay.UserDataRelayOutputPacket</pre>
sort() (digi.xbee.reader.ModemStatusReceived	attribute), 373
method), 976	ssid (digi.xbee.models.accesspoint.AccessPoint at-
<pre>sort() (digi.xbee.reader.NetworkModified method),</pre>	tribute), 125
978	start() (digi.xbee.reader.PacketListener method),
sort () (digi.xbee.reader.PacketReceived method), 973	1001
sort() (digi.xbee.reader.PacketReceivedFrom method),	start_discovery_process()
974  (diai who a mader Polan Pata Possined mothed)	(digi.xbee.devices.DigiMeshNetwork method),
sort() (digi.xbee.reader.RelayDataReceived method), 983	918 start_discovery_process()
sort() (digi.xbee.reader.RouteInformationReceived	(digi.xbee.devices.DigiPointNetwork method),
method), 989	929
sort () (digi.xbee.reader.RouteReceived method), 990	start_discovery_process()
sort() (digi.xbee.reader.RouteRecordIndicatorReceived	(digi.xbee.devices.Raw802Network method),
method), 988	907
sort () (digi.xbee.reader.SMSReceived method), 982	start_discovery_process()
sort() (digi.xbee.reader.SocketDataReceived method),	(digi.xbee.devices.XBeeNetwork method),
986	874
sort() (digi.xbee.reader.SocketDataReceivedFrom	start_discovery_process()

(digi.xbee.device 895	es.ZigBeeNetwork method),	status	attribute), 162 (digi.xbee.models.filesystem.OpenFileCmdRequest
start_listening()		beacab	attribute), 139
	es.CellularDevice method),	status	(digi.xbee.models.filesystem.OpenFileCmdResponse attribute), 141
<pre>start_listening()     method),644</pre>	(digi.xbee.devices.IPDevice	status	(digi.xbee.models.filesystem.ReadDirCmdRequest attribute), 167
start_listening() (digi.xbee.device	es.LPWANDevice method),	status	(digi.xbee.models.filesystem.ReadDirCmdResponse attribute), 169
720 start_listening()			(digi.xbee.models.filesystem.ReadFileCmdRequest attribute), 146
(digi.xbee.device 746			(digi.xbee.models.filesystem.ReadFileCmdResponse attribute), 148
method), 779	(digi.xbee.devices.WiFiDevice		(digi.xbee.models.filesystem.RenameCmdRequest attribute), 175
state (digi.xbee.packet.	info.SocketInfo attribute), 191 s.socket.SocketStatePacket at-		(digi.xbee.models.filesystem.RenameCmdResponse attribute), 176
	tem.FileProcess attribute), 938	status	attribute), 134
attribute), 131	-		(digi.xbee.models.filesystem.VolFormatCmdRequest attribute), 184
attribute), 164			(digi.xbee.models.filesystem.VolFormatCmdResponse attribute), 186
attribute), 165			(digi.xbee.models.filesystem.VolStatCmdRequest attribute), 181
attribute), 143			(digi.xbee.models.filesystem.VolStatCmdResponse attribute), 183
attribute), 144			(digi.xbee.models.filesystem.WriteFileCmdRequest attribute), 150
attribute), 157			(digi.xbee.models.filesystem.WriteFileCmdResponse attribute), 152
attribute), 158			(digi.xbee.models.zdo.Route attribute), 232 (digi.xbee.packets.common.ATCommResponsePacket
attribute), 178	s.filesystem.DeleteCmdRequest		attribute), 261 (digi.xbee.packets.common.RemoteATCommandResponsePacket
attribute), 179	s.filesystem.DeleteCmdResponse		attribute), 273 (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket
tribute), 136	lels.filesystem.FileIdCmd at-	status	attribute), 307 (digi.xbee.packets.devicecloud.SendDataResponsePacket
attribute), 137	dels.filesystem.FileIdNameCmd	status	attribute), 318 (digi.xbee.packets.socket.SocketCloseResponsePacket
132	ds.filesystem.FSCmd attribute),		attribute), 398 (digi.xbee.packets.socket.SocketConnectResponsePacket
status ( <i>digi.xbee.models</i> attribute), 171	s.filesystem.GetPathIdCmdReque		attribute), 392 (digi.xbee.packets.socket.SocketCreateResponsePacket
status (digi.xbee.models attribute), 173	s.filesystem.GetPathIdCmdRespo		attribute), 378 (digi.xbee.packets.socket.SocketListenResponsePacket
status (digi.xbee.models attribute), 154	s.filesystem.HashFileCmdReques		attribute), 411 (digi.xbee.packets.socket.SocketOptionResponsePacket
	s.file system. Hash File CmdRe sport	nse	attribute), 385 (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket
	s.filesystem.OpenDirCmdReques	st	attribute), 436 (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
	s.filesystem.OpenDirCmdRespon		attribute), 441

```
status_a2b (digi.xbee.devices.Connection attribute), status_value(digi.xbee.models.filesystem.UnknownFSCmd
                                                              attribute), 134
         933
status b2a (digi.xbee.devices.Connection attribute), status value (digi.xbee.models.filesystem.VolFormatCmdRequest
        934
                                                              attribute), 184
status_value(digi.xbee.models.filesystem.CloseDirCmdReaguest_value(digi.xbee.models.filesystem.VolFormatCmdResponse
        attribute), 164
                                                              attribute), 186
status value (digi.xbee.models.filesystem.CloseDirCmdResponsevalue (digi.xbee.models.filesystem.VolStatCmdRequest
        attribute), 166
                                                               attribute), 181
status value (digi.xbee.models.filesystem.CloseFileCmdRequest value (digi.xbee.models.filesystem.VolStatCmdResponse
        attribute), 143
                                                              attribute), 183
status_value(digi.xbee.models.filesystem.CloseFileCmdResponse_value(digi.xbee.models.filesystem.WriteFileCmdRequest
        attribute), 144
                                                              attribute), 150
status_value(digi.xbee.models.filesystem.CreateDirCmxRetpuest_value(digi.xbee.models.filesystem.WriteFileCmdResponse
        attribute), 157
                                                               attribute), 152
status_value(digi.xbee.models.filesystem.CreateDirCmdRasp/on&digi.xbee.models.zdo.NeighborFinder method),
        attribute), 159
status_value(digi.xbee.models.filesystem.DeleteCmdRequest()
                                                                 (digi.xbee.models.zdo.NeighborTableReader
        attribute), 178
                                                              method), 233
status_value(digi.xbee.models.filesystem.DeleteCmdRestponste)
                                                                (digi.xbee.models.zdo.NodeDescriptorReader
        attribute), 180
                                                              method), 227
status_value(digi.xbee.models.filesystem.FileIdCmd stop()
                                                                    (digi.xbee.models.zdo.RouteTableReader
                                                              method), 231
        attribute), 136
status_value(digi.xbee.models.filesystem.FileIdNameCsrtdp() (digi.xbee.reader.PacketListener method), 993
        attribute), 138
                                                      stop bits (digi.xbee.profile.FirmwareStopbits at-
                                                              tribute), 964
status value
                   (digi.xbee.models.filesystem.FSCmd
        attribute), 132
                                                     stop_discovery_process()
status_value(digi.xbee.models.filesystem.GetPathIdCmdRequest(digi.xbee.devices.DigiMeshNetwork method),
        attribute), 171
status_value(digi.xbee.models.filesystem.GetPathIdCmdResportisescovery_process()
        attribute), 173
                                                               (digi.xbee.devices.DigiPointNetwork method),
status_value(digi.xbee.models.filesystem.HashFileCmdRequest 930
        attribute), 154
                                                      stop_discovery_process()
status_value(digi.xbee.models.filesystem.HashFileCmdResponse(digi.xbee.devices.Raw802Network
                                                                                                 method),
        attribute), 155
status_value(digi.xbee.models.filesystem.OpenDirCmdReqpestiscovery_process()
                                                              (digi.xbee.devices.XBeeNetwork
        attribute), 160
                                                                                                 method),
status value (digi.xbee.models.filesystem.OpenDirCmdResponse875
        attribute), 162
                                                      stop_discovery_process()
status_value(digi.xbee.models.filesystem.OpenFileCmdRequest (digi.xbee.devices.ZigBeeNetwork
                                                                                                 method),
                                                               896
        attribute), 140
status value (digi.xbee.models.filesystem.OpenFileCmdResponsestening() (digi.xbee.devices.CellularDevice
        attribute), 141
                                                              method), 694
status_value(digi.xbee.models.filesystem.ReadDirCmdReapes1 istening()
                                                                                (digi.xbee.devices.IPDevice
        attribute), 167
                                                              method), 644
status_value(digi.xbee.models.filesystem.ReadDirCmd&espon%astening()(digi.xbee.devices.LPWANDevice
        attribute), 169
                                                               method), 720
status_value(digi.xbee.models.filesystem.ReadFileCmdReappestistening() (digi.xbee.devices.NBIoTDevice
        attribute), 146
                                                              method), 746
status_value(digi.xbee.models.filesystem.ReadFileCmdResponsicstening()
                                                                              (digi.xbee.devices.WiFiDevice
        attribute), 148
                                                              method), 779
status_value(digi.xbee.models.filesystem.RenameCmdRequestssor_addr(digi.xbee.packets.digimesh.RouteInformationPacket
        attribute), 175
                                                              attribute), 323
status_value(digi.xbee.models.filesystem.RenameCmdResppossets_apply_profile()
        attribute), 176
                                                               (digi.xbee.comm interface.XBeeCommunicationInterface
```

```
method), 466
                                                                                                                                                                                                                                                             to dict()
                                                                                                                                                                                                                                                                                                                                         (digi.xbee.models.filesystem.FileIdCmd
supports_apply_profile()
                                                                                                                                                                                                                                                                                                       method), 136
                                          (digi.xbee.serial.XBeeSerialPort
                                                                                                                                                                                                            method),
                                                                                                                                                                                                                                                            to dict()(digi.xbee.models.filesystem.FileIdNameCmd
                                                                                                                                                                                                                                                                                                       method), 138
supports_update_firmware()
                                                                                                                                                                                                                                                             to dict()
                                                                                                                                                                                                                                                                                                                                                       (digi.xbee.models.filesystem.FSCmd
                                         (digi.xbee.comm interface.XBeeCommunicationInterface method), 133
                                         method), 465
                                                                                                                                                                                                                                                             to dict()(digi.xbee.models.filesystem.GetPathIdCmdRequest
supports update firmware()
                                                                                                                                                                                                                                                                                                       method), 171
                                                                                                                                                                                                                                                            to\_dict() (digi.xbee.models.filesystem.GetPathIdCmdResponse
                                          (digi.xbee.serial.XBeeSerialPort
                                                                                                                                                                                                            method).
                                                                                                                                                                                                                                                                                                       method), 173
sync_sleep (digi.xbee.firmware.UpdateConfigurer at-
                                                                                                                                                                                                                                                            to\_dict() (digi.xbee.models.filesystem.HashFileCmdRequest
                                          tribute), 954
                                                                                                                                                                                                                                                                                                       method), 154
                                                                                                                                                                                                                                                            \verb+to_dict()+ (\textit{digi.xbee.models.filesystem.HashFileCmdResponse})
SyncRequestSender (class in digi.xbee.sender),
                                                                                                                                                                                                                                                                                                       method), 156
                                          1005
                                                                                                                                                                                                                                                             to_dict() (digi.xbee.models.filesystem.OpenDirCmdRequest
                                                                                                                                                                                                                                                                                                       method), 161
                                                                                                                                                                                                                                                           to_dict() (digi.xbee.models.filesystem.OpenDirCmdResponse
tag (digi.xbee.profile.XBeeSettingFormat attribute), 966
tag (digi.xbee.profile.XBeeSettingType attribute), 965
                                                                                                                                                                                                                                                                                                       method), 163
target(digi.xbee.packets.devicecloud.DeviceRequestPacktp_dict()(digi.xbee.models.filesystem.OpenFileCmdRequest
                                          attribute), 301
                                                                                                                                                                                                                                                                                                       method), 140
task_done() (digi.xbee.reader.XBeeQueue method),
                                                                                                                                                                                                                                                            to_dict() (digi.xbee.models.filesystem.OpenFileCmdResponse
                                                                                                                                                                                                                                                                                                       method), 142
                                          1003
\verb|timeout| (\textit{digi.xbee.comm\_interface.XBeeCommunication} | \textit{facet}() | (\textit{digi.xbee.models.filesystem.ReadDirCmdRequest}) | \textit{timeout}() | (\textit{digi.xbee.models.filesystem.ReadDirCmdRequest}) | \textit{timeout}() | \textit{tim
                                                                                                                                                                                                                                                                                                       method), 167
                                          attribute), 466
timeout
                                                          (digi.xbee.sender.SyncRequestSender
                                                                                                                                                                                                                                                            to_dict() (digi.xbee.models.filesystem.ReadDirCmdResponse
                                          tribute), 1005
                                                                                                                                                                                                                                                                                                       method), 169
                                                                                                                                                                                                                                                            to_dict() (digi.xbee.models.filesystem.ReadFileCmdRequest
timeout (digi.xbee.serial.XBeeSerialPort attribute),
                                          1009
                                                                                                                                                                                                                                                                                                       method), 147
                                                                                                                                                                                                                                                             to_dict() (digi.xbee.models.filesystem.ReadFileCmdResponse
TIMEOUT READ PACKET
                                                                                                                                                                                                                                                                                                       method), 148
                                          (digi.xbee.devices.XBeeDevice
                                                                                                                                                                                                       attribute),
                                         485
                                                                                                                                                                                                                                                             to_dict() (digi.xbee.models.filesystem.RenameCmdRequest
                                                                                                                                                                                                                                                                                                       method), 175
TimeoutException, 936
timestamp(digi.xbee.models.message.ExplicitXBeeMessage_dict() (digi.xbee.models.filesystem.RenameCmdResponse
                                                                                                                                                                                                                                                                                                       method), 177
                                          attribute), 201
method), 134
                                         attribute), 199
method), 185
                                          attribute), 320
\verb|to_dict()| (\textit{digi.xbee.models.filesystem.CloseDirCmdReqtuest} \\ \texttt{dict()}| (\textit{digi.xbee.models.filesystem.VolFormatCmdResponse}) \\ \\ \texttt{dict()}| (\textit{digi.xbee.models.filesystem.Vol
                                                                                                                                                                                                                                                                                                       method), 187
                                         method), 164
\verb|to_dict()| (digi.xbee.models.filesystem.CloseDirCmdResponselict()| (digi.xbee.models.filesystem.VolStatCmdRequest)| (digi.xbee.models.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.f
                                                                                                                                                                                                                                                                                                       method), 181
                                          method), 166
\verb|to_dict()| (\textit{digi.xbee.models.filesystem.CloseFileCmdRedpass} | \textit{dict()}| (\textit{digi.xbee.models.filesystem.VolStatCmdResponse}|) | \textit{digi.xbee.models.filesystem.VolStatCmdResponse}| | \textit{d
                                                                                                                                                                                                                                                                                                       method), 183
                                          method), 143
\verb|to_dict()| (digi.xbee.models.filesystem.CloseFileCmdResponseict()| (digi.xbee.models.filesystem.WriteFileCmdRequest)| (digi.xbee.models.filesystem.WriteFileCmdRequest)| (digi.xbee.models.filesystem.WriteFileCmdRequest)| (digi.xbee.models.filesystem.WriteFileCmdResponseict())| (digi.xbee.modelsystem.WriteFileCmdResponseict())| (digi.xbee.modelsystem.WriteFileCmdResponseict())| (digi.xbee
                                                                                                                                                                                                                                                                                                       method), 150
                                          method), 145
to_dict() (digi.xbee.models.filesystem.CreateDirCmdRetaesxlict() (digi.xbee.models.filesystem.WriteFileCmdResponse
                                         method), 157
                                                                                                                                                                                                                                                                                                       method), 152
\verb|to_dict()| (digi.xbee.models.filesystem.CreateDirCmdRetponSect()| (digi.xbee.models.message.ExplicitXBeeMessage)| (digi.xbee.models.filesystem.CreateDirCmdRetponSect()| (di
                                                                                                                                                                                                                                                                                                       method), 200
                                         method), 159
                                                                                                                                                                                                                                                                                                                                             (digi.xbee.models.message.IPMessage
to_dict() (digi.xbee.models.filesystem.DeleteCmdRequesto_dict()
                                          method), 178
                                                                                                                                                                                                                                                                                                       method), 202
to_dict() (digi.xbee.models.filesystem.DeleteCmdRespo#se_dict()
                                                                                                                                                                                                                                                                                                                                   (digi.xbee.models.message.SMSMessage
                                          method), 180
                                                                                                                                                                                                                                                                                                       method), 202
```

- to\_dict() (digi.xbee.models.message.UserDataRelayMetsagedict() (digi.xbee.packets.filesystem.FSRequestPacket method), 203 method), 326
- to\_dict() (digi.xbee.models.message.XBeeMessage to\_dict() (digi.xbee.packets.filesystem.FSResponsePacket method), 199 method), 328
- to\_dict() (digi.xbee.packets.base.GenericXBeePacket to\_dict() (digi.xbee.packets.filesystem.RemoteFSRequestPacket method), 244 method), 332
- to\_dict() (digi.xbee.packets.base.UnknownXBeePacket to\_dict() (digi.xbee.packets.filesystem.RemoteFSResponsePacket method), 247 method), 334
- to\_dict() (digi.xbee.packets.base.XBeeAPIPacket to\_dict() (digi.xbee.packets.network.RXIPv4Packet method), 242 method), 339
- to\_dict() (digi.xbee.packets.base.XBeePacket to\_dict() (digi.xbee.packets.network.TXIPv4Packet method), 239 method), 342
- to\_dict() (digi.xbee.packets.cellular.RXSMSPacket to\_dict() (digi.xbee.packets.raw.RX16IOPacket method), 250 method), 365
- to\_dict() (digi.xbee.packets.cellular.TXSMSPacket to\_dict() (digi.xbee.packets.raw.RX16Packet method), 253 method), 359
- to\_dict() (digi.xbee.packets.common.ATCommPacket to\_dict() (digi.xbee.packets.raw.RX64IOPacket method), 256 method), 363
- to\_dict() (digi.xbee.packets.common.ATCommQueuePacket\_dict() (digi.xbee.packets.raw.RX64Packet method), 259 method), 356
- to\_dict() (digi.xbee.packets.common.ATCommResponsetPacketct() (digi.xbee.packets.raw.TX16Packet method), 263 method), 349
- to\_dict() (digi.xbee.packets.common.ExplicitAddressingPacketct() (digi.xbee.packets.raw.TX64Packet method), 293 method), 346
- to\_dict() (digi.xbee.packets.common.ExplicitRXIndicatorPacket() (digi.xbee.packets.raw.TXStatusPacket method), 297 method), 352
- to\_dict() (digi.xbee.packets.common.IODataSampleRxIndicatorRacketdigi.xbee.packets.relay.UserDataRelayOutputPacket method), 290 method), 373
- to\_dict() (digi.xbee.packets.common.ModemStatusPacketo\_dict() (digi.xbee.packets.relay.UserDataRelayPacket method), 286 method), 370
- to\_dict() (digi.xbee.packets.common.ReceivePacket to\_dict() (digi.xbee.packets.socket.SocketBindListenPacket method), 267 method), 409
- to\_dict() (digi.xbee.packets.common.RemoteATCommatudPackett() (digi.xbee.packets.socket.SocketClosePacket method), 270 method), 396
- to\_dict() (digi.xbee.packets.common.RemoteATCommandResportseRaddiegi.xbee.packets.socket.SocketCloseResponsePacket method), 275 method), 400
- to\_dict() (digi.xbee.packets.common.TransmitPacket to\_dict() (digi.xbee.packets.socket.SocketConnectPacket method), 390
- to\_dict() (digi.xbee.packets.common.TransmitStatusPacket\_dict() (digi.xbee.packets.socket.SocketConnectResponsePacket method), 283 method), 393
- to\_dict() (digi.xbee.packets.devicecloud.DeviceRequestPacketict() (digi.xbee.packets.socket.SocketCreatePacket method), 302 method), 376
- to\_dict() (digi.xbee.packets.devicecloud.DeviceResponseBacketct() (digi.xbee.packets.socket.SocketCreateResponsePacket method), 305 method), 379
- to\_dict() (digi.xbee.packets.devicecloud.DeviceResponseStattisPacket(digi.xbee.packets.socket.SocketListenResponsePacket method), 308 method), 412
- to\_dict() (digi.xbee.packets.devicecloud.FrameErrorPacket\_dict() (digi.xbee.packets.socket.SocketNewIPv4ClientPacket method), 311 method), 416
- to\_dict() (digi.xbee.packets.devicecloud.SendDataRequ**extPacket**() (digi.xbee.packets.socket.SocketOptionRequestPacket method), 315 method), 383
- to\_dict() (digi.xbee.packets.devicecloud.SendDataResponsePacket () (digi.xbee.packets.socket.SocketOptionResponsePacket method), 317 method), 386
- $\verb|to_dict()| (digi.xbee.packets.digimesh.RouteInformation \textit{Pack} \textit{etict()} (digi.xbee.packets.socket.SocketReceiveFromPacket method), 322 \\ method), 421$

```
to dict()(digi.xbee.packets.socket.SocketReceivePacketTransmitStatusPacket
                                                                                           (class
                                                                                                         in
         method), 418
                                                                digi.xbee.packets.common), 279
to_dict() (digi.xbee.packets.socket.SocketSendPacket transport(digi.xbee.packets.devicecloud.DeviceRequestPacket
                                                                attribute), 301
        method), 403
to_dict() (digi.xbee.packets.socket.SocketSendToPacket TRUNCATE (digi.xbee.models.options.FileOpenRequestOption
        method), 406
                                                               attribute), 211
to dict() (digi.xbee.packets.socket.SocketStatePacket TX16Packet (class in digi.xbee.packets.raw), 346
                                                      TX64Packet (class in digi.xbee.packets.raw), 343
         method), 424
to_dict() (digi.xbee.packets.wifi.IODataSampleRxIndicaterWifiBacketount (digi.xbee.packets.digimesh.RouteInformationPacket
         method), 429
                                                                attribute), 320
to_dict() (digi.xbee.packets.wifi.RemoteATCommandResponseANtfileacket (class in digi.xbee.packets.network),
         method), 435
                                                                339
to_dict() (digi.xbee.packets.wifi.RemoteATCommandWiftRankePacket (class in digi.xbee.packets.cellular), 250
        method), 432
                                                      TXStatusPacket (class in digi.xbee.packets.raw),
to\_dict() (digi.xbee.packets.zigbee.CreateSourceRoutePacket
                                                                350
         method), 451
                                                       type (digi.xbee.models.filesystem.CloseDirCmdRequest
to_dict() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacketibute), 164
        method), 453
                                                      type (digi.xbee.models.filesystem.CloseDirCmdResponse
to_dict() (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket attribute), 166
        method), 443
                                                      type (digi.xbee.models.filesystem.CloseFileCmdRequest
to_dict() (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket attribute), 143
        method), 440
                                                      type(digi.xbee.models.filesystem.CloseFileCmdResponse)
to_dict() (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket attribute), 145
         method), 447
                                                      type (digi.xbee.models.filesystem.CreateDirCmdRequest
transmit options (digi.xbee.packets.common.ExplicitAddressingPaidhete), 158
        attribute), 294
                                                      type (digi.xbee.models.filesystem.CreateDirCmdResponse
transmit_options (digi.xbee.packets.common.RemoteATCommandflboke), 159
                                                      type(digi.xbee.models.filesystem.DeleteCmdRequest\ at-
        attribute), 269
transmit_options (digi.xbee.packets.common.TransmitPacket tribute), 178
        attribute), 277
                                                      type (digi.xbee.models.filesystem.DeleteCmdResponse
transmit_options (digi.xbee.packets.filesystem.RemoteFSRequest#Pribkete), 180
                                                      type (digi.xbee.models.filesystem.FileIdCmd attribute),
        attribute), 330
transmit_options (digi.xbee.packets.network.TXIPv4Packet
                                                                136
                                                       type (digi.xbee.models.filesystem.FileIdNameCmd at-
        attribute), 343
transmit_options (digi.xbee.packets.raw.TX16Packet
                                                                tribute), 138
                                                      type (digi.xbee.models.filesystem.FSCmd attribute), 132
        attribute), 348
transmit_options (digi.xbee.packets.raw.TX64Packet type (digi.xbee.models.filesystem.GetPathIdCmdRequest
        attribute), 344
                                                                attribute), 172
transmit_options (digi.xbee.packets.wifi.RemoteATCompoundMifiRbekanodels.filesystem.GetPathIdCmdResponse
        attribute), 431
                                                                attribute), 173
                                                      type (digi.xbee.models.filesystem.HashFileCmdRequest
transmit retry count
                                                                attribute), 154
         (digi.xbee.packets.common.TransmitStatusPacket
                                                      type(digi.xbee.models.filesystem.HashFileCmdResponse)
        attribute), 281
transmit_status(digi.xbee.packets.common.TransmitStatusPackettribute), 156
                                                      type (digi.xbee.models.filesystem.OpenDirCmdRequest
        attribute), 281
transmit_status(digi.xbee.packets.raw.TXStatusPacket
                                                                attribute), 161
                                                      type (digi.xbee.models.filesystem.OpenDirCmdResponse
        attribute), 351
TransmitException, 936
                                                                attribute), 163
                                                     type (digi.xbee.models.filesystem.OpenFileCmdRequest
TransmitOptions
                                 (class
        digi.xbee.models.options), 204
                                                                attribute), 140
                                                      type(digi.xbee.models.filesystem.OpenFileCmdResponse)
TransmitPacket
                                (class
                                                                attribute), 142
        digi.xbee.packets.common), 275
TransmitStatus (class in digi.xbee.models.status), type (digi.xbee.models.filesystem.ReadDirCmdRequest
         215
                                                                attribute), 167
```

type (digi.xbee.models.filesystem.ReadDirCmdResponse unescape\_data() (digi.xbee.packets.common.ReceivePacket static method), 267 attribute), 169 type (digi.xbee.models.filesystem.ReadFileCmdRequest unescape\_data() (digi.xbee.packets.common.RemoteATCommandPack attribute), 147 static method), 271  ${\tt type} \ (\textit{digi.xbee.models.filesystem.ReadFileCmdResponse} \quad {\tt unescape\_data()} \ (\textit{digi.xbee.packets.common.RemoteATCommandResponse})$ attribute), 149 static method), 275 type (digi.xbee.models.filesystem.RenameCmdRequest unescape\_data() (digi.xbee.packets.common.TransmitPacket attribute), 175 static method), 279  $\verb|type| (digi.xbee.models.filesystem.RenameCmdResponse|$ unescape\_data() (digi.xbee.packets.common.TransmitStatusPacket attribute), 177 static method), 283 type (digi.xbee.models.filesystem.UnknownFSCmd atunescape\_data() (digi.xbee.packets.devicecloud.DeviceRequestPacket tribute), 133 static method), 302 type(digi.xbee.models.filesystem.VolFormatCmdRequest unescape\_data()(digi.xbee.packets.devicecloud.DeviceResponsePack attribute), 185 static method), 306  ${\tt type} \ (digi.xbee.models.filesystem.VolFormatCmdResponse \verb"unescape_data" () \ (digi.xbee.packets.devicecloud.DeviceResponseStatus) \ (digi.xbee.packets.devicecloud.Devic$ attribute), 187 static method), 309 (digi.xbee.models.filesystem.VolStatCmdRequest unescape\_data() (digi.xbee.packets.devicecloud.FrameErrorPacket static method), 311 attribute), 182 type (digi.xbee.models.filesystem.VolStatCmdResponse unescape\_data() (digi.xbee.packets.devicecloud.SendDataRequestPac attribute), 183 static method), 315 type (digi.xbee.models.filesystem.WriteFileCmdRequest unescape\_data() (digi.xbee.packets.devicecloud.SendDataResponsePac attribute), 151 static method), 317  $type(digi.xbee.models.filesystem.WriteFileCmdResponse\ unescape\_data()(digi.xbee.packets.digimesh.RouteInformationPackets)$ attribute), 152 static method), 322 (digi.xbee.profile.XBeeProfileSetting attribute), unescape\_data() (digi.xbee.packets.filesystem.FSRequestPacket 966 static method), 326 unescape\_data()(digi.xbee.packets.filesystem.FSResponsePacket U static method), 329 unescape\_data() (digi.xbee.packets.base.GenericXBeePacketape\_data() (digi.xbee.packets.filesystem.RemoteFSRequestPacket static method), 332 static method), 244 unescape\_data() (digi.xbee.packets.base.UnknownXBeePackeape\_data() (digi.xbee.packets.filesystem.RemoteFSResponsePackeape\_data() static method), 334 static method), 247 unescape\_data() (digi.xbee.packets.base.XBeeAPIPacketnescape\_data() (digi.xbee.packets.network.RXIPv4Packet static method), 339 static method), 242 unescape\_data() (digi.xbee.packets.base.XBeePacket unescape\_data() (digi.xbee.packets.network.TXIPv4Packet static method), 342 static method), 239 unescape\_data() (digi.xbee.packets.cellular.RXSMSPackeescape\_data() (digi.xbee.packets.raw.RX16IOPacket static method), 365 static method), 250 unescape\_data() (digi.xbee.packets.cellular.TXSMSPackeescape\_data() (digi.xbee.packets.raw.RX16Packet static method), 360 static method), 253 unescape\_data()(digi.xbee.packets.common.ATCommPackerape\_data()(digi.xbee.packets.raw.RX64IOPacket static method), 256 static method), 363 unescape\_data() (digi.xbee.packets.common.ATCommQuenePapket\_data() (digi.xbee.packets.raw.RX64Packet static method), 356 static method), 259 unescape\_data()(digi.xbee.packets.common.ATCommResponsePacketa()(digi.xbee.packets.raw.TX16Packet static method), 349 static method), 263 unescape\_data() (digi.xbee.packets.common.ExplicitAddressingPacketa() (digi.xbee.packets.raw.TX64Packet static method), 293 static method), 346  $\verb"unescape_data" () \textit{ (digi.xbee.packets.common.ExplicitRXIndicatopPacket} \texttt{a} \texttt{()} \textit{ (digi.xbee.packets.raw.TXS} \textit{tatusPacket} \texttt{)}$ static method), 352 static method), 297 unescape\_data() (digi.xbee.packets.common.IODataSampleRelpdicatorPacketdigi.xbee.packets.relay.UserDataRelayOutputPacke static method), 373

1102 Index

unescape\_data() (digi.xbee.packets.common.ModemStatusPackpe\_data() (digi.xbee.packets.relay.UserDataRelayPacket

static method), 370

static method), 290

static method), 286

```
unescape_data() (digi.xbee.packets.socket.SocketBindListenPackættribute), 932
        static method), 409
                                                     UnknownFSCmd (class in digi.xbee.models.filesystem),
unescape data() (digi.xbee.packets.socket.SocketClosePacket 133
                                                     UnknownXBeePacket
        static method), 396
                                                                                      (class
                                                                                                     in
unescape_data() (digi.xbee.packets.socket.SocketCloseResponseHigikxbee.packets.base), 245
        static method), 400
                                                    unregister joining device()
unescape data() (digi.xbee.packets.socket.SocketConnectPacket(digi.xbee.devices.ZigBeeDevice
                                                                                               method),
        static method), 390
unescape_data()(digi.xbee.packets.socket.SocketConnectRespiratePacketining_device_async()
        static method), 394
                                                             (digi.xbee.devices.ZigBeeDevice
                                                                                               method),
unescape_data()(digi.xbee.packets.socket.SocketCreatePacket 614
        static method), 376
                                                     update_bluetooth_password()
unescape_data() (digi.xbee.packets.socket.SocketCreateResponseProjetee.devices.AbstractXBeeDevice
        static method), 380
                                                             method), 483
unescape_data() (digi.xbee.packets.socket.SocketListemRasportse_Backettooth_password()
        static method), 412
                                                              (digi.xbee.devices.CellularDevice
                                                                                               method),
unescape_data()(digi.xbee.packets.socket.SocketNewIPv4ClientPacket
        static method), 416
                                                    update bluetooth password()
unescape_data() (digi.xbee.packets.socket.SocketOptionRequestPdigiexbee.devices.DigiMeshDevice
                                                                                               method),
        static method), 383
unescape_data() (digi.xbee.packets.socket.SocketOptionResponsetooth_password()
        static method), 387
                                                             (digi.xbee.devices.DigiPointDevice
                                                                                               method),
unescape_data() (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                     update bluetooth password()
        static method), 421
unescape_data() (digi.xbee.packets.socket.SocketReceivePacket (digi.xbee.devices.IPDevice method), 668
        static method), 418
                                                     update bluetooth password()
unescape_data() (digi.xbee.packets.socket.SocketSendPacket
                                                             (digi.xbee.devices.LPWANDevice
                                                                                               method),
        static method), 403
unescape_data() (digi.xbee.packets.socket.SocketSendToPacket_bluetooth_password()
        static method), 406
                                                              (digi.xbee.devices.NBIoTDevice
                                                                                               method),
unescape_data() (digi.xbee.packets.socket.SocketStatePacket
                                                             746
        static method), 424
                                                     update_bluetooth_password()
unescape_data() (digi.xbee.packets.wifi.IODataSampleRxIndicatehiXifibeecktevices.Raw802Device
                                                                                               method),
        static method), 429
unescape data() (digi.xbee.packets.wifi.RemoteATCommandRespin) neeWifiDatketassword()
        static method), 435
                                                              (digi.xbee.devices.RemoteDigiMeshDevice
unescape data() (digi.xbee.packets.wifi.RemoteATCommandWifiPacket), 833
        static method), 433
                                                    update_bluetooth_password()
unescape_data() (digi.xbee.packets.zigbee.CreateSourceRoutePacketi.xbee.devices.RemoteDigiPointDevice
        static method), 451
                                                             method), 851
unescape data() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusePacketh password()
        static method), 453
                                                              (digi.xbee.devices.RemoteRaw802Device
unescape_data() (digi.xbee.packets.zigbee.RegisterDeviceStatusPredbetd), 815
        static method), 443
                                                     update_bluetooth_password()
unescape_data() (digi.xbee.packets.zigbee.RegisterJoiningDevicedHagikettee.devices.RemoteXBeeDevice
        static method), 440
                                                              method), 797
unescape_data() (digi.xbee.packets.zigbee.RouteRecordfulicatentPacketooth_password()
        static method), 447
                                                             (digi.xbee.devices.RemoteZigBeeDevice
UNKNOWN (digi.xbee.devices.LinkQuality attribute), 932
                                                             method), 871
UNKNOWN_ADDRESS (digi.xbee.models.address.XBee16BitApdress._bluetooth_password()
                                                              (digi.xbee.devices.WiFiDevice method), 779
        attribute), 196
UNKNOWN ADDRESS (digi.xbee.models.address.XBee64BitAddrests. bluetooth password()
        attribute), 197
                                                              (digi.xbee.devices.XBeeDevice
                                                                                               method),
UNKNOWN VALUE
                       (digi.xbee.devices.LinkQuality
                                                              510
```

```
method), 852
update_bluetooth_password()
        (digi.xbee.devices.ZigBeeDevice
                                                  update_filesystem_image()
                                         method).
                                                           (digi.xbee.devices.RemoteRaw802Device
                                                           method), 815
update_device_data_from()
        (digi.xbee.devices.AbstractXBeeDevice
                                                  update_filesystem_image()
        method), 467
                                                           (digi.xbee.devices.RemoteXBeeDevice
update_device_data_from()
                                                           method), 782
        (digi.xbee.devices.CellularDevice
                                                  update_filesystem_image()
                                         method),
        695
                                                           (digi.xbee.devices.RemoteZigBeeDevice
update_device_data_from()
                                                           method), 872
        (digi.xbee.devices.DigiMeshDevice
                                         method),
                                                  update_firmware()
                                                           (digi.xbee.comm_interface.XBeeCommunicationInterface
update_device_data_from()
                                                           method), 465
        (digi.xbee.devices.DigiPointDevice
                                                  update_firmware()
                                         method),
                                                           (digi.xbee.devices.AbstractXBeeDevice
update_device_data_from()
                                                           method), 483
        (digi.xbee.devices.IPDevice method), 668
                                                  update_firmware()
update_device_data_from()
                                                           (digi.xbee.devices.CellularDevice
                                                                                           method),
        (digi.xbee.devices.LPWANDevice
                                                           695
                                         method),
                                                  update firmware()
update_device_data_from()
                                                           (digi.xbee.devices.DigiMeshDevice
                                                                                           method),
        (digi.xbee.devices.NBIoTDevice
                                         method),
        746
                                                  update_firmware()
update_device_data_from()
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                           method),
        (digi.xbee.devices.Raw802Device
                                                           604
                                         method),
                                                  update_firmware()
                                                                           (digi.xbee.devices.IPDevice
update_device_data_from()
                                                           method), 668
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                  update_firmware()
        method), 834
                                                           (digi.xbee.devices.LPWANDevice
                                                                                           method),
                                                           721
update_device_data_from()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                  update_firmware()
        method), 852
                                                           (digi.xbee.devices.NBIoTDevice
                                                                                           method),
                                                           746
update_device_data_from()
        (digi.xbee.devices.RemoteRaw802Device
                                                  update_firmware()
        method), 815
                                                           (digi.xbee.devices.Raw802Device
                                                                                           method),
update_device_data_from()
        (digi.xbee.devices.RemoteXBeeDevice
                                                  update_firmware()
        method), 797
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
update_device_data_from()
                                                           method), 834
        (digi.xbee.devices.RemoteZigBeeDevice
                                                  update_firmware()
        method), 872
                                                           (digi.xbee.devices.RemoteDigiPointDevice
update_device_data_from()
                                                           method), 852
        (digi.xbee.devices.WiFiDevice method), 779
                                                  update_firmware()
update_device_data_from()
                                                           (digi.xbee.devices.RemoteRaw802Device
                                                           method), 816
        (digi.xbee.devices.XBeeDevice
                                         method),
        510
                                                  update_firmware()
update_device_data_from()
                                                           (digi.xbee.devices.RemoteXBeeDevice
        (digi.xbee.devices.ZigBeeDevice
                                                           method), 797
                                         method),
        641
                                                  update_firmware()
                                                           (digi.xbee.devices.RemoteZigBeeDevice
update_filesystem_image()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                           method), 872
        method), 834
                                                  update_firmware() (digi.xbee.devices.WiFiDevice
update_filesystem_image()
                                                           method), 779
        (digi.xbee.devices.RemoteDigiPointDevice
                                                  update_firmware() (digi.xbee.devices.XBeeDevice
```

method), 511	W
<pre>update_firmware()</pre>	wait_for_frame()(digi.xbee.comm_interface.XBeeCommunicationIr
(digi.xbee.devices.ZigBeeDevice method), 641	method), 464
update_firmware()	wait_for_frame() (digi.xbee.serial.XBeeSerialPort
(digi.xbee.serial.XBeeSerialPort method),	method), 1007
1009	<pre>wait_until_started()</pre>
update_local_firmware() (in module	(digi.xbee.reader.PacketListener method), 993
digi.xbee.firmware), 956	WiFiAssociationIndicationStatus (class in
update_remote_filesystem() (in module digi.xbee.firmware), 956	digi.xbee.models.status), 222
<pre>update_remote_filesystem_image() (in mod-</pre>	WiFiDevice (class in digi.xbee.devices), 747
ule digi.xbee.filesystem), 953	WiFiEncryptionType (class in digi.xbee.models.accesspoint), 125
update_remote_firmware() (in module digi.xbee.firmware), 956	<pre>with_traceback() (digi.xbee.exception.ATCommandException</pre>
UpdateConfigurer (class in digi.xbee.firmware),	with_traceback() (digi.xbee.exception.CommunicationException
954 UpdateProfileException, 967	method), 934
_ <del>-</del>	<pre>with_traceback() (digi.xbee.exception.ConnectionException</pre>
USE_BROADCAST_PAN_ID	method), 935
(digi.xbee.models.options.TransmitOptions	<pre>with_traceback() (digi.xbee.exception.FirmwareUpdateException</pre>
attribute), 205	method), 936
USE_CURRENT_OFFSET	with_traceback() (digi.xbee.exception.InvalidConfigurationException
(digi.xbee.models.filesystem.ReadFileCmdReque	memoa), >33
attribute), 145	with_traceback() (digi.xbee.exception.InvalidOperatingModeException.
USE_CURRENT_OFFSET	method), 935
(atgi.xbee.modets.filesystem.writeFileCmaReque attribute), 149	estwith_traceback() (digi.xbee.exception.InvalidPacketException method), 935
USE_EXTENDED_TIMEOUT	with_traceback() (digi.xbee.exception.OperationNotSupportedExceptionNotSupportedExceptio
(digi.x bee.models.options. Transmit Options	method), 936
attribute), 206	with_traceback() (digi.xbee.exception.RecoveryException
user_desc_supported	method), 936
(digi.xbee.models.zdo.NodeDescriptor at-	with_traceback() (digi.xbee.exception.TimeoutException
tribute), 228	method), 936
UserDataRelayMessage (class in	with_traceback() (digi.xbee.exception.TransmitException
digi.xbee.models.message), 202	method), 936
UserDataRelayOutputPacket (class in	with_traceback() (digi.xbee.exception.XBeeDeviceException
digi.xbee.packets.relay), 370	method), 935
UserDataRelayPacket (class in	with_traceback() (digi.xbee.exception.XBeeException
digi.xbee.packets.relay), 367	method), 934
	with_traceback() (digi.xbee.exception.XBeeSocketException
V	method), 936
value (digi.xbee.profile.XBeeProfileSetting attribute),	with_traceback() (digi.xbee.filesystem.FileSystemException
967	method), 938
version (digi.xbee.profile.XBeeProfile attribute), 968	with_traceback() (digi.xbee.filesystem.FileSystemNotSupportedExce
VolFormatCmdRequest (class in	method), 938
digi.xbee.models.filesystem), 184	<pre>with_traceback() (digi.xbee.profile.ReadProfileException</pre>
VolFormatCmdResponse (class in	method), 967
digi.xbee.models.filesystem), 185	<pre>with_traceback() (digi.xbee.profile.UpdateProfileException</pre>
VolStatCmdRequest (class in	method), 967
digi.xbee.models.filesystem), 180	with_traceback() (digi.xbee.util.xmodem.XModemCancelException
VolStatCmdResponse (class in	method), 462
digi.xbee.models.filesystem), 182	<pre>with_traceback() (digi.xbee.util.xmodem.XModemException</pre>
	method), 462
	WRITE (digi.xbee.models.options.FileOpenRequestOption

```
attribute), 210
                                                                                                                   x16bit_dest_addr(digi.xbee.packets.raw.TX16Packet
write_changes() (digi.xbee.devices.AbstractXBeeDevice
                                                                                                                                      attribute), 348
                                                                                                                   x16bit_dest_addr(digi.xbee.packets.zigbee.CreateSourceRoutePacke
                  method), 469
write_changes() (digi.xbee.devices.CellularDevice
                                                                                                                                      attribute), 449
                  method), 695
                                                                                                                  x16bit source addr
write_changes() (digi.xbee.devices.DigiMeshDevice
                                                                                                                                      (digi.xbee.packets.common. Explicit RXIndicator Packet
                  method), 573
                                                                                                                                      attribute), 298
write_changes() (digi.xbee.devices.DigiPointDevice x16bit_source_addr
                                                                                                                                      (digi.xbee.packets.common.IODataSampleRxIndicatorPacket
                   method), 604
write_changes()
                                                         (digi.xbee.devices.IPDevice
                                                                                                                                      attribute), 288
                  method), 669
                                                                                                                   x16bit_source_addr
write_changes() (digi.xbee.devices.LPWANDevice
                                                                                                                                      (digi.xbee.packets.common.ReceivePacket
                  method), 721
                                                                                                                                      attribute), 265
                                                (digi.xbee.devices.NBIoTDevice
                                                                                                                  x16bit_source_addr
write_changes()
                                                                                                                                      (digi.x bee.packets.common.Remote ATC ommand Response Packet\\
                   method), 747
write_changes() (digi.xbee.devices.Raw802Device
                                                                                                                                      attribute), 273
                   method), 541
                                                                                                                   x16bit_source_addr
write_changes() (digi.xbee.devices.RemoteDigiMeshDevice
                                                                                                                                      (digi.xbee.packets.raw.RX16IOPacket
                                                                                                                                                                                                                            at-
                  method), 835
                                                                                                                                      tribute), 366
write_changes() (digi.xbee.devices.RemoteDigiPointDevi6toit_source_addr
                  method), 853
                                                                                                                                      (digi.xbee.packets.raw.RX16Packet attribute),
write_changes() (digi.xbee.devices.RemoteRaw802Device
                   method), 816
                                                                                                                   x16bit_source_addr
                                                                                                                                      (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
write_changes() (digi.xbee.devices.RemoteXBeeDevice
                  method), 798
                                                                                                                                      attribute), 445
write_changes()(digi.xbee.devices.RemoteZigBeeDevict6bit_updater_addr
                   method), 873
                                                                                                                                      (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
write_changes()
                                                    (digi.xbee.devices.WiFiDevice
                                                                                                                                      attribute), 454
                  method), 780
                                                                                                                   x64bit_dest_addr(digi.xbee.packets.common.ExplicitAddressingPack
write_changes()
                                                   (digi.xbee.devices.XBeeDevice
                                                                                                                                      attribute), 294
                   method), 511
                                                                                                                   x64bit_dest_addr(digi.xbee.packets.common.RemoteATCommandPa
write_changes()
                                               (digi.xbee.devices.ZigBeeDevice
                                                                                                                                      attribute), 268
                   method), 642
                                                                                                                  x64bit_dest_addr(digi.xbee.packets.common.TransmitPacket
                                                                                                                                      attribute), 277
write_file() (digi.xbee.filesystem.FileSystemManager
                   method), 941
                                                                                                                   x64bit_dest_addr (digi.xbee.packets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.RemoteFSRequestPackets.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesystem.filesys
write_frame() (digi.xbee.comm_interface.XBeeCommunicationInterface), 330
                  method), 465
                                                                                                                   x64bit_dest_addr(digi.xbee.packets.raw.TX64Packet
write_frame()
                                                (digi.xbee.serial.XBeeSerialPort
                                                                                                                                      attribute), 344
                  method), 1006
                                                                                                                   x64bit_dest_addr(digi.xbee.packets.zigbee.CreateSourceRoutePacke
WriteFileCmdRequest
                                                                          (class
                                                                                                          in
                                                                                                                                      attribute), 449
                   digi.xbee.models.filesystem), 149
                                                                                                                  x64bit source addr
WriteFileCmdResponse
                                                                                                                                      (digi.xbee.packets.common.ExplicitRXIndicatorPacket
                                                                            (class
                                                                                                          in
                  digi.xbee.models.filesystem), 151
                                                                                                                                      attribute), 298
                                                                                                                   x64bit_source_addr
X
                                                                                                                                      (digi.xbee.packets.common.IOD ataSample RxIndicator Packet
x16bit_dest_addr(digi.xbee.packets.common.ExplicitAddressingPailNete), 287
                                                                                                                  x64bit_source_addr
                   attribute), 295
\verb|x16bit_dest_addr| (\textit{digi.xbee.packets.common.RemoteATCommon.RemoteATCommon.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceivePackets.common.ReceiveP
                                                                                                                                      attribute), 265
                   attribute), 269
x16bit_dest_addr(digi.xbee.packets.common.Transmixfacket_source_addr
                                                                                                                                      (digi.xbee.packets.common.RemoteATCommandResponsePacket
                   attribute), 277
x16bit_dest_addr(digi.xbee.packets.common.TransmitStatusPackttibute), 273
                                                                                                                  x64bit_source_addr
                   attribute), 281
                                                                                                                                      (digi.xbee.packets.filesystem.RemoteFSResponsePacket
```

```
attribute), 335
                                                     ZigbeeRegisterStatus
                                                                                        (class
                                                                                                      in
x64bit_source_addr
                                                              digi.xbee.models.status), 223
        (digi.xbee.packets.raw.RX64IOPacket
        tribute), 361
x64bit source addr
        (digi.xbee.packets.raw.RX64Packet attribute),
         354
x64bit source addr
         (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
        attribute), 454
x64bit_source_addr
        (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
        attribute), 445
x64bit_target_addr
        (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
        attribute), 455
        (digi.xbee.filesystem.FileSystemManager
xbee
         tribute), 939
xbee
       (digi.xbee.sender.SyncRequestSender attribute),
         1005
XBee16BitAddress
                                (class
                                                 in
        digi.xbee.models.address), 195
XBee64BitAddress
                                (class
                                                 in
        digi.xbee.models.address), 197
XBeeAPIPacket (class in digi.xbee.packets.base), 240
XBeeCommunicationInterface
                                       (class
                                                 in
        digi.xbee.comm_interface), 464
XBeeDevice (class in digi.xbee.devices), 484
XBeeDeviceException, 935
XBeeEvent (class in digi.xbee.reader), 971
XBeeException, 934
XBeeIMEIAddress
                                (class
                                                 in
        digi.xbee.models.address), 198
XBeeLocalInterface
                                 (class
                                                 in
        digi.xbee.models.options), 208
XBeeMessage (class in digi.xbee.models.message), 199
XBeeNetwork (class in digi.xbee.devices), 873
XBeePacket (class in digi.xbee.packets.base), 238
XBeeProfile (class in digi.xbee.profile), 967
XBeeProfileSetting (class in digi.xbee.profile),
XBeeProtocol (class in digi.xbee.models.protocol),
XBeeQueue (class in digi.xbee.reader), 1001
XBeeSerialPort (class in digi.xbee.serial), 1006
XBeeSettingFormat (class in digi.xbee.profile), 965
XBeeSettingType (class in digi.xbee.profile), 965
XBeeSocketException, 936
XModemCancelException, 462
XModemException, 461
Ζ
```

ZigBeeDevice (class in digi.xbee.devices), 605 ZigBeeNetwork (class in digi.xbee.devices), 885