# **XBee Python Library Documentation**

Release 1.4.1

Digi International Inc.

## Getting Started

1	Requ	iiremer	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	API re	eference.		6
		2.6.1	Get sta	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	AT settings or commands	15
			2.6.2.6	Radio module operating modes	16
			2.6.2.7	API frames	18
		2.6.3		vith XBee classes	18
			2.6.3.1	Instantiate an XBee object	19
			2.6.3.2	Open the XBee connection	21
			2.6.3.3	Close the XBee connection	24
		2.6.4	_	ure the XBee	25
			2.6.4.1	Read and set common parameters	25
			2.6.4.2	Read, set and execute other parameters	27
			2.6.4.3	Apply configuration changes	30
			2.6.4.4	Write configuration changes	31
			2.6.4.5	Reset the device	31
			2.6.4.6	Configure Wi-Fi settings	32
		265	2.6.4.7	Configure Bluetooth settings	34
		2.6.5		er the XBee network	36
			2.6.5.1	Discovery types	37
			2.6.5.2	Deep discovery	37

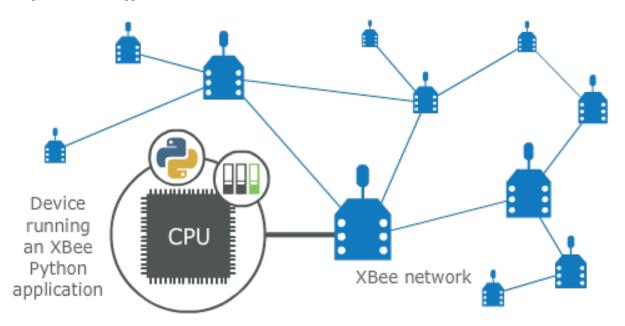
	2.6.5.3 2.6.5.4 2.6.5.5 2.6.5.6	Standard discovery Discover the network Access discovered nodes Access connections between nodes	. 40 . 44 . 45
2.6.6	2.6.5.7 2.6.5.8 Commu 2.6.6.1	Add and remove nodes manually	. 48 . 50
	2.6.6.2 2.6.6.3 2.6.6.4	Send and receive explicit data	. 57 . 64 . 70
	2.6.6.5 2.6.6.6 2.6.6.7 2.6.6.8	Send and receive Bluetooth data Send and receive MicroPython data Receive modem status events Communicate using XBee sockets	. 75 . 77
2.6.7	Handle 2.6.7.1 2.6.7.2 2.6.7.3	analog and digital IO lines	. 82 . 87
2.6.8	Update 2.6.8.1 2.6.8.2 2.6.8.3	the XBee  Update the XBee firmware  Update the XBee file system  Apply an XBee profile	. 92 . 92 . 97
2.6.9 2.6.10	2.6.9.1	Update multiple nodes	. 108 . 110
2.0.10	2.6.10.1 2.6.10.2 2.6.10.3 2.6.10.4 2.6.10.5 2.6.10.6	Configuration samples  Network samples  Communication samples  IO samples  Firmware samples  File system samples  Profile samples	. 111 . 112 . 113 . 119 . 120 . 121
2.6.11	Frequer 2.6.11.1 2.6.11.2 2.6.11.3 2.6.11.4	the third Asked Questions (FAQs)	. 122 . 122 . 122 . 123 . 123
2.6.12	Change 2.6.12.1 2.6.12.2 2.6.12.3 2.6.12.4 2.6.12.5	I get the Python error ImportError: No module named 'srp'  v1.4.1 - 12/22/2021  v1.4.0 - 03/18/2021  v1.3.0 - 11/05/2019  v1.2.0 - 04/05/2018  1.1.0 01/10/2018	<ul><li>. 124</li><li>. 124</li><li>. 125</li><li>. 126</li><li>. 127</li><li>. 127</li></ul>
2.6.13 <b>Indices and</b>	2.6.12.7 API ref 2.6.13.1	v1.1.0 - 01/19/2018	. 128 . 129
License			1043

Python Module Index	1045
Index	1045

Release v1.4.1. (*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
- 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 them.

#### 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.

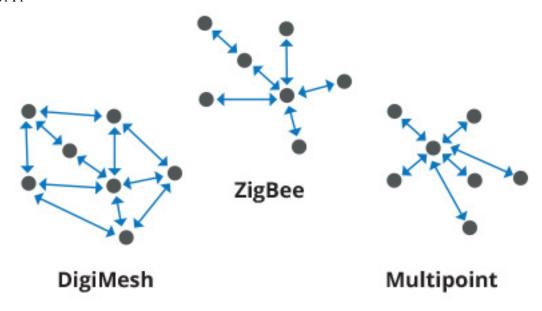
To update or change the firmware of the local XBee module or any other module operating in the same network, use the mechanisms the XBee Python Library includes. Other programs, such as XCTU, or the web interface of the XBee Gateway, also allows you to update the firmware of your XBee nodes.

#### 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
- Wi-Fi



**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 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 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**. When working in this mode, entering in command mode will also allow the configuration of the local XBee. But to configure or execute AT commands in API mode, generate an AT command API frame containing the AT setting and the value of that setting, and send it to the XBee RF module. For more information about API mode see, see *API operating mode*.

#### 2.6.2.6 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 (AP=2) or not (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 (AP=0), API (AP=1) or API escaped (AP=2).

#### **Application Transparent (AT) operating mode**

In Application Transparent (AT) or transparent operating mode, all data received through the serial input is queued up for radio transmission and data received wirelessly is sent to the serial output exactly as it is received. In fact, communication in transparent mode yields the same result as if the two modules were connected by a wire, but wireless communication makes that physical wire unnecessary.

Some advantages of this mode:

- XBee in transparent mode act as a serial line replacement: what you send is exactly what the other module get.
- It is compatible with any device that speaks serial.
- It works very well when facilitating communication between two XBees.

Transparent mode has some limitations. For example:

- When working with several remote nodes, you must configure the destination before sending each message.
- It is not possible to identify the source of a received wireless message.
- To access the configuration of an XBee in transparent mode a special procedure for transitioning the module into *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 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.
- Advanced network management and diagnosis.
- Advanced features such as remote firmware update, ZDO, ZCL, etc.

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

### API escaped operating mode

API escaped operating mode (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: Escape

• 0x11: XON

• 0x13: XOFF

#### **Command mode**

Command mode allows to get and set local XBee parameters and execute certain AT commands.

To enter command mode, send the 3-character command sequence through the serial interface of the radio module, usually +++, within one second. Once the XBee is operating in command mode, the module sends the reply OK, the command mode timer starts, and the data coming from the serial input is interpreted as commands to set up the module.

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).

#### 2.6.2.7 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	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	

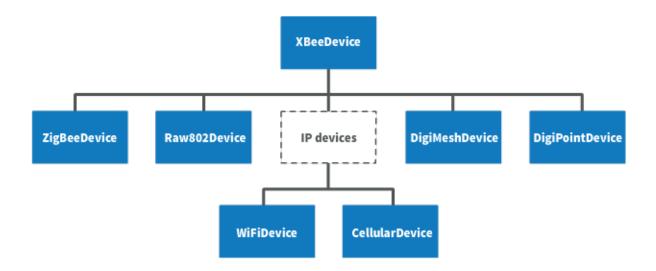
In API escaped mode, some bytes in the Length, Frame Data and Checksum fields must be escaped.



#### 2.6.3 Work with XBee classes

When working with the XBee Python Library, start with an XBee object that represents a physical module. A physical XBee 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.

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



- XBee Zigbee (ZigBeeDevice)
- XBee 802.15.4 (Raw802Device)
- XBee DigiMesh (DigiMeshDevice)
- XBee Point-to-multipoint (DigiPointDevice)
- XBee IP devices (This is a non-instantiable class)
  - XBee Cellular (CellularDevice)
  - XBee Wi-Fi (WiFiDevice)

All these XBee classes allow you to configure the physical XBee, 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, instantiate a generic XBeeDevice object or one that is protocol-specific.

**Note:** This documentation refers to the XBeeDevice object when describing the different features, but they are also applicable to any XBee protocol-specific class.

#### 2.6.3.1 Instantiate an XBee object

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

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

#### Local XBee node

A local XBee is the object representing the device 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, provide the following two parameters:

- · Serial port name
- · Serial port baud rate

#### Instantiate a local XBee

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

#### Remote XBee node

Remote XBee 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 must provide a local XBee that operates in the same network and 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 a remote device cannot send data to another remote device.

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

• Local XBee 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 attached to your PC and the **16-bit address** of the remote device.

#### Instantiate a remote XBee

```
xbee = XBeeDevice("COM1", 9600)
remote = RemoteXBeeDevice(xbee, XBee64BitAddress.from_hex_string("0013A20012345678"))
[...]
```

**Note:** Local and remote devices must use the same protocol.

#### 2.6.3.2 Open the XBee connection

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

Remote XBee devices do not have an equivalent method. They use a local XBee as the connection interface. To perform any operation with a remote XBee, open the connection of the associated local device.

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

# Open the device connection.
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.

This method also configures the operating mode of the local XBee to API mode without escapes (AP=1) if its not using an API mode (AP=1 or AP=2)

```
[ · · · ] (continues on next page)
```

(continued from previous page)

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

# Open the connection using constructor parameters: 9600 8N1.
# This reconfigures the XBee if its serial settings do not match.
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/RecoverSerialConnection/RecoverSerialConnection.py

#### Read device information

The read device information process gets some relevant data from the local or remote XBee and stores it. Once cached, you can access this information at any time, calling the corresponding getter. This process reads the following data:

- 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 automatically performed in local XBee devices when opening them with the open() method. Remote XBee devices cannot be opened, use read\_device\_info() to read their device information.

#### Initialize a remote XBee

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

# Instantiate a remote XBee device object.
remote = RemoteXBeeDevice(xbee, XBee64BitAddress.from_hex_string("0013A20040XXXXXX"))

# Read the device information of the remote XBee.
remote.read_device_info()

[...]
```

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

- ACK of the sent command 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()

# Get the 64-bit address of the device.
addr_64 = xbee.get_64bit_addr()
# Get the node identifier of the device.
node_id = xbee.get_node_id()
# Get the hardware version of the device.
hardware_version = xbee.get_hardware_version()
# Get the firmware version of the device.
firmware_version = xbee.get_firmware_version()
```

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

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

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

#### Get the XBee protocol

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

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

#### **Device operating mode**

The open () process also reads the operating mode of the physical local XBee 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

```
[...] (continues on next page)
```

(continued from previous page)

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

# Get the operating mode of the device.
operating_mode = 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.

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

- API
- · API with escaped characters

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

**Note:** If you are not sure of the operating mode of your local XBee, use force\_settings=True as parameter of open() method. This reconfigures the XBee serial settings (baud rate, data bits, stop bits, etc.) to those specified in the XBee object constructor, including the operating mode of the XBee to be API (AP=1) if its not already using an API mode.

#### 2.6.3.3 Close the XBee connection

Call the close () method when you finish working with the local XBee. For example, before exiting your application.

This method guarantees the serial port where your XBee is connected will not be used for any operation and will remain close.

#### Close the connection

```
[...]
# Instantiate a local XBee object.
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, close the connection of the local associated device.

### 2.6.4 Configure the XBee

One of the 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 these changes in the device. For more information, see *Write configuration changes*.

#### 2.6.4.1 Read and set common parameters

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

#### **Cached parameters**

Certain XBee parameters are used or requested frequently. To avoid the overhead of reading them from the physical XBee every time they are requested, their values are cached 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 their values.

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

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

### Refresh cached parameters

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Refresh the cached values.
```

(continues on next page)

(continued from previous page)

```
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 (NI) do not change; therefore, they cannot be set. For the Node Identifier, there is a method within all the XBee classes that allows you to change it:

Method	Description
set_node_id(	<b>String</b> ) fies the new Node Identifier of the device. This method configures the physical XBee 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 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 (that is, IO sampling data). This setting can be read and set.

Method	Description	
get_dest_address()	Returns the 64-bit address of the device that data will be reported to.	
set_dest_address(XBee64BitAddress) Specifies the 64-bit address of the device where the data will be		
	ported.	

• PAN ID: This is the ID of the Personal Area Network the XBee 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	
	is operating.	
set_pan_id(Bytearray)Specifies the value in byte array format of the PAN ID where the XBee should work.		

• Power level: This setting specifies the output power level of the XBee. 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.	
set_power_level(PowerLevel) pecifies a PowerLevel enumeration entry containing the desired output level		
	of the XBee.	

#### Configure non-cached parameters

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

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

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

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

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

[...]
```

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

- ACK of the sent command 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

You can read or set a parameter that does not have a custom getter or setter within the XBee object. All the XBee 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.

#### Get a parameter

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

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

You can also use get\_parameter () for settings with a specific getter in the API.

#### Get a parameter from the XBee

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Get the value of the Sleep Time (SP) parameter.
sp = xbee.get_parameter("SP")
[...]
```

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

- ACK of the sent command 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, use the set\_parameter() method provided by all the XBee classes.

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

You can also use set\_parameter() for settings with a specific setter in the API.

#### Set a parameter in the XBee

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

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

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

- ACK of the sent command 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. The XBee Python Library has several commands that handle the most common executable parameters. 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 classes.

Method	Description	
<b>execute_command(String)</b> Specifies the AT command (String format) to be run i		

#### Run a command in the XBee

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Run the apply changes command.
xbee.execute_command("AC")
[...]
```

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

- ACK of the sent command 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 <code>XBeeException</code>.

#### 2.6.4.3 Apply configuration changes

By default, when you perform any configuration on a local or remote XBee, the changes are automatically applied. However, you may want to configure different settings or parameters of a device and apply these changes at the same time. For that purpose, the XBeeDevice and RemoteXBeeDevice objects provide some methods to manage when to apply configuration changes.

Method	Description	Notes
en-	Specifies whether the changes on settings	The apply configuration changes flag is en-
able_apply_changes(Roodeam) eters are applied when set.		abled by default.
is_apply_changes_enableds()whether the XBee is configured to		
	apply parameter changes when they are set.	
<b>apply_changes</b> () Applies parameters changes that were al-		This method is useful when the XBee is con-
	ready set but are pending to be applied.	figured not to apply changes when they are set.

#### **Apply configuration changes**

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Check if device is configured to apply changes.
apply_changes_enabled = xbee.is_apply_changes_enabled()
# Configure the device not to apply parameter changes automatically.
if apply_changes_enabled:
    xbee.enable_apply_changes (False)
# Set the PAN ID of the XBee to BABE.
xbee.set_pan_id(utils.hex_string_to_bytes("BABE"))
# Perform other configurations.
[...]
# Apply changes.
xbee.apply_changes()
[...]
```

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

- ACK of the sent command 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  ${\tt XBeeException}.$

# 2.6.4.4 Write configuration changes

For the configuration changes performed in an XBee to persist through subsequent resets, save those changes. Saving changes means that configured parameter values in the device are written to the non-volatile memory of the XBee. The module loads these values from non-volatile memory every time it is started.

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

# Write configuration changes

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

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

# Perform other configurations.
[...]

# Apply changes.
xbee.apply_changes()

# Write changes.
xbee.write_changes()
```

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

- ACK of the sent command 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 when the system is not operating properly or you are initializing the system. All the XBee 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 a local XBee object.

(continues on next page)
```

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

# Reset the module.
xbee.reset()

[...]
```

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

- ACK of the sent command 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. 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 Wi-Fi object.
xbee = WiFiDevice("COM1", 9600)
xbee.open()
# Configure the IP addressing mode to DHCP.
xbee.set_ip_addressing_mode(IPAddressingMode.DHCP)
# Save the IP addressing mode.
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**

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 Wi-Fi object.
xbee = WiFiDevice("COM1", 9600)
xbee.open()

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

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

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

You can also change these 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 XBee 3 devices have a Bluetooth® Low Energy (BLE) interface that enables you to connect your XBee to another device such as a cellphone. The XBee 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 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.
disable_bluetooth()	Disables the Bluetooth Low Energy interface of your XBee.

# Enabling and disabling the Bluetooth interface

```
[...]
# Instantiate a local XBee object.
```

(continues on next page)

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

# Enable the Bluetooth interface.
xbee.enable_bluetooth()

[...]

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

These methods may fail for the following reasons:

- ACK of the sent command 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 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.

# Configuring or changing the Bluetooth password

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
new_password = "myBluetoothPassword" # Do not hard-code it in the app!
# Configure the Bluetooth password.
xbee.update_bluetooth_password(new_password)
[...]
```

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

- ACK of the sent command 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

The XBee Java Library provides the get\_bluetooth\_mac\_addr() method to return the EUI-48 Bluetooth MAC address of your XBee following the format "00112233AABB".

# Reading the Bluetooth MAC address

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

The get\_bluetooth\_mac\_addr() method may fail for the following reasons:

- ACK of the sent command 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 <code>XBeeException</code>.

# 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, to represent 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().

# 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)	<ul> <li>Configures the deep discovery timeouts:         <ul> <li>node_timeout (Float, optional): Maximum duration in seconds of the discovery process used to find neighbors of a node.</li> <li>time_bw_requests (Float, optional): Time to wait between node neighbors requests (in seconds)</li></ul></li></ul>
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 val-
	ues that are applied to the local XBee module when per-
	forming the discovery process. These options are the
	following:
	• <b>DiscoveryOptions.APPEND_DD</b> : Appends the
	device type identifier (DD) to the information re-
	trieved when a node is discovered. This option
	is valid for DigiMesh, Point-to-multipoint (Digi
	Point) and Zigbee protocols.
	<ul><li>DiscoveryOptions.DISCOVER_MYSELF:</li></ul>
	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

# 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.
	• deep (Boolean, optional): True for a deep net-
	work scan, False otherwise. See <i>Discovery</i>
	types.
	• n_deep_scans (Integer, optional): Number of
	discovery scans to perform. Only for deep dis-
	covery.

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 <code>get\_devices()</code> method provided by the network object. If the discovery process is running, this method returns <code>None</code>.

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.
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, 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):
    [...]
# 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, 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

The XBeeNetwork object also provides methods to discover specific nodes within a network. This may be useful, for example, 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	(Strqings) 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-	<b>dis-</b> Specify the node identifiers of the XBee nodes to find. Returns a list with the remote XBee nodes	
cover_devices	s([Strong]))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	4RittArds ness) 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(XBee1	get_device_by_16(XBee16RittArhdress)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	get_device_by_node_id(\$tRing) rns 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"
```

(continues on next page)

```
# 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(AbstractxxBeeDevioe)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()
```

(continues on next page)

```
# 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	add_remoteXiBeeDeXide) to add 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	ld_rem&tes([RembteXiBeeDXVIce])hodes 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(...)

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

(continues on next page)

```
# 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	move_device(Remote XB ceDeviice) 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, i		
	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 to perform a clean discovery, deleting the list before calling the discovery method.

# Methodescription 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 Python 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, provide a callback using the  $add\_network\_modified\_callback()$  method. This callback must follow the format:

```
def my_net_modified_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:")
                  Type: %s (%d)" % (event_type.description, event_type.code))
   print("
   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)
[...]
```

To stop listening to network modifications, use the del\_network\_modified\_callback() method to unsubscribe the already-registered callback.

# Deregister a network modification callback

```
def cb_network_modified(event_type, reason, node):
        [...]

xbee.add_network_modified_callback(cb_network_modified)

[...]

# Delete the callback.
xbee.del_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.PROFILE\_UPDATE: The node was updated/removed, or the network cleared after applying a profile.
- 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 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 and any remote on the network.

- Send data
- Receive data

# Send data

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

You can send data either using a unicast or a 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	<b>d Seas</b> ; ifies 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

```
[...]
# Instantiate a local XBee object.

(continues on next page)
```

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

# Instantiate a remote XBee object.
remote = RemoteXBeeDevice(xbee, XBee64BitAddress.from_hex_string("0013A20040XXXXXX"))

# Send data using the remote object.
xbee.send_data(remote, "Hello XBee!")

[...]
```

The previous methods may fail for the following reasons:

- ACK of the sent command 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 class.

### Get/set the timeout for synchronous operations

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

xbee = [...]
# Retrieving the configured timeout for synchronous operations.
print("Current timeout: %d seconds" % xbee.get_sync_ops_timeout())

[...]
# Configuring the new timeout (in seconds) for synchronous operations.
xbee.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 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 node.

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>is pdd fress</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	2B <b>end</b> edata_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>dapess</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>is pdd fress</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

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

# Instantiate a remote XBee object.
remote = RemoteXBeeDevice(xbee, XBee64BitAddress.from_hex_string("0013A20040XXXXXX"))

# Send data using the remote object.
xbee.send_data_async(remote, "Hello XBee!")

[...]
```

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 asynchronously. The example is located in the following path:

examples/communication/SendData AsyncSample

# Send data to all devices of the network

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

All the XBee 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Send broadcast data.
xbee.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 (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. This method blocks your application until data from any XBee on the network is received or the provided timeout expires:

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 (polling)

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Read data.
xbee_message = xbee.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 = xbee.read_data()

remote = 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 of the network. For that purpose, the XBee object provides the read\_data\_from() method:

Method	Description	
read_data_from (RpmoffeXBreeDewite, XBee to read data from and the time to wait for data reception (method		
Integer)	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 (polling)

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Instantiate a remote XBee object.
remote = RemoteXBeeDevice(xbee, XBee64BitAddress.from_hex_string("0013A200XXXXXX"))
# Read data sent by the remote device.
xbee_message = xbee.read_data(remote)
[...]
```

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 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.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.
xbee.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):
    [...]

xbee.add_data_received_callback(my_data_received_callback)
[...]

# Delete the callback
xbee.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 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 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 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 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(RemoteXBeeDevi	<b>c</b> specifies remote XBee destination object, four application layer fields
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/SendExplicitDataAsyncSample

# 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 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 local Zigbee object.
xbee = ZigBeeDevice("COM1", 9600)
xbee.open()
# Send broadcast data.
xbee.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, configure the data output mode of the receiver XBee to explicit format using the set\_api\_output\_mode\_value() method.

Method	Description	
get_api_output_mode	<b>Relue()</b> the API output mode of the data received by the XBee.	
set_api_output_mode_Salue(Enterger)PI output mode of the data received by the XBee. 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. 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	<b>Depta(Mateghe)</b> 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 (polling)

```
[...]
# Instantiate a local Zigbee object.
xbee = ZigBeeDevice("COM1", 9600)
xbee.open()
# Read data.
xbee_message = xbee.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 = xbee.read_expl_data()

remote = 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 of the network. For that purpose, the XBee object provides the read\_expl\_data\_from() method:

Method	Description	
read_expl_data_from and the time to wait for explicit data		
Integer)	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 (polling)

```
[...]
# Instantiate a local Zigbee object.
xbee = ZigBeeDevice("COM1", 9600)
xbee.open()
# Instantiate a remote Zigbee object.
remote = RemoteZigBeeDevice(xbee, XBee64BitAddress.from_hex_string("0013A200XXXXXX"))
# Read data sent by the remote device.
expl_xbee_message = xbee.read_expl_data(remote)
[...]
```

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 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 local Zigbee object.
xbee = ZigBeeDevice("COM1", 9600)
xbee.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.
xbee.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):
    [...]

xbee.add_expl_data_received_callback(my_expl_data_received_callback)
```

2.6. API reference 63

(continues on next page)

```
[...]
# Delete the callback
xbee.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: 0x0011 - Profile ID: 0xC105

• For remote IO sample:

- Source endpoint: 0xE8

- Destination endpoint: 0xE8

- Cluster ID: 0x0092 - Profile 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 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 an XBee Cellular 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 an XBee Cellular 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 an XBee Cellular object.
xbee = CellularDevice("COM1", 9600)
xbee.open()

# Listen for TCP or UDP transmissions at port 1234.
xbee.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 an XBee Cellular object.
xbee = CellularDevice("COM1", 9600)
xbee.open()
```

(continues on next page)

```
# Stop listening for TCP or UDP transmissions.
xbee.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	tas[peteiges]) 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 an XBee Cellular object.
xbee = CellularDevice("COM1", 9600)
xbee.open()

# Read IP data.
ip_message = xbee.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 an XBee Cellular object.
```

(continues on next page)

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

# Read IP data.
ip_message = xbee.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 an XBee Cellular object.
xbee = CellularDevice("COM1", 9600)
xbee.open()

# Read IP data.
ip_message = xbee.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 an XBee Cellular object.
xbee = CellularDevice("COM1", 9600)
xbee.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.
xbee.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

```
[...]
xbee = [...]

def my_ip_data_received_callback(ip_message):
        [...]

xbee.add_ip_data_received_callback(my_ip_data_received_callback)

[...]

# Delete the IP data callback.
xbee.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 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 an XBee Cellular 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 an XBee Cellular 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 an XBee Cellular object.
xbee CellularDevice("COM1", 9600)
xbee.open()
```

(continues on next page)

```
# 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.
xbee.add_sms_callback(my_sms_callback)

[...]
```

When a new SMS message is received, your callback is executed providing an SMSMessage object as parameter. 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**

```
[...]
xbee = [...]

def my_sms_callback(sms_message):
    [...]

xbee.add_sms_callback(my_sms_callback)

[...]

# Delete the SMS callback.
xbee.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

XBee 3 modules have the ability to send and receive data from the Bluetooth Low Energy interface of the local XBee 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 XBee 3 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()

data = "Bluetooth, are you there?"

# Send the data to the Bluetooth interface.
xbee.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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()

# Define the callback.
def my_bluetooth_data_callback(data):
```

(continues on next page)

```
print("Data received from the Bluetooth interface >> '%s'" % data.decode("utf-8"))
# Add the callback.
xbee.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**

```
[...]
xbee = [...]

def my_bluetooth_data_callback(data):
    [...]
xbee.add_bluetooth_data_received_callback(my_bluetooth_data_callback)
[...]

# Delete the Bluetooth data callback.
xbee.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

XBee 3 modules have the ability to send and receive data from the MicroPython interface of the local XBee 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 XBee 3 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()

data = "MicroPython, are you there?"

# Send the data to the MicroPython interface.
xbee.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

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

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

# Add the callback.
xbee.add_micropython_data_received_callback(my_micropython_data_callback)
```

(continues on next page)

```
[...]
```

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**

```
[...]
xbee = [...]

def my_micropython_data_callback(data):
    [...]

xbee.add_micropython_data_received_callback(my_micropython_data_callback)
[...]

# Delete the MicroPython data callback.
xbee.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 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, 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()

# Define the callback.
def my_modem_status_callback(status):
```

(continues on next page)

```
print("Modem status: %s" % status.description)

# Add the callback.
xbee.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

```
[...]
xbee = [...]

def my_modem_status_callback(status):
    [...]

xbee.add_modem_status_received_callback(my_modem_status_callback)
[...]

# Delete the modem status callback.
xbee.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 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, provide the following parameters:

- XBee Cellular 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.
xbee = CellularDevice("COM1", 9600)
xbee.open()

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

#### Work with an XBee socket

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
	Connects to a remote socket at the provided address. The address must be a pair (host, port),
	where <i>host</i> is the domain name or string representation of an IPv4 and <i>port</i> is the numeric port value.
	Closes the socket.
	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(Bytearra	<b>S</b> ends the provided data to the socket. The socket must be connected to a remote socket.
sendto(Bytear	Sany 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)).
	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.
	<b>ger</b> eives 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)).
_	Returns the value of the provided socket option.
opt(SocketOp	•
	Sets the value of the provided socket option.
opt(SocketOp	tion,
Bytear-	
ray)	
_	Returns the configured socket timeout in seconds.
out()	
	Sets the socket timeout in seconds.
out(Integer)	
_	Returns whether the socket is in blocking mode or not.
ing()	
	Sets the socket in blocking or non-blocking mode. In blocking mode, operations block until com-
	plete or the system returns an error. In non-blocking mode, operations fail if they cannot be com-
	pleted within the configured timeout.
	Returns the information of the socket, including the socket ID, state, protocol, local port, remote
	port and remote address.
	tAtelscallbank(Eudctillh)ck to be notified when a new socket state is received.
del socket sta	adelead back (Fundation) eket 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.
xbee = CellularDevice("COM1", 9600)
xbee.open()

# Create a new XBee socket.
with xsocket.socket(xbee, 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.
xbee = CellularDevice("COM1", 9600)
xbee.open()
# Create a new XBee socket.
with xsocket.socket(xbee, 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 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 you are connecting.

**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 objects include a configuration method, set\_io\_configuration(), to specify the IO line to configure and their desired function.

For the IO line parameter, the API provides an enumerator called IOLine that helps you specify the desired IO line 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Instantiate a remote XBee object.
remote = RemoteXBeeDevice(xbee, XBee64BitAddress.from_hex_string("0013A20012345678"))
# Configure the DIO1_AD1 line of the local device to be Digital output (set high by,
⇔default).
xbee.set_io_configuration(IOLine.DIO1_AD1, IOMode.DIGITAL_OUT_HIGH)
# Configure the DIO2_AD2 line of the local device to be Digital input.
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.set_io_configuration(IOLine.DIO3_AD3, IOMode.ADC)
# Configure the DIO10_PWM0 line of the remote device to be PWM output (PWM).
remote.set_io_configuration(IOLine.DIO10_PWM0, IOMode.PWM)
[...]
```

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

- ACK of the sent command 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 using the corresponding getter, get\_io\_configuration().

#### **Get IO configuration**

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

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

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

- ACK of the sent command 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). All the XBee classes provide the method set\_dio\_value(), with the desired IOLine as the first parameter and an IOValue as the second one. The IOValue enumerator includes HIGH and LOW as possible values.

#### Set digital output values

```
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
# Set the DIO2_AD2 line low.
xbee.set_dio_value(IOLine.DIO2_AD2, IOValue.LOW)
# Set the DIO2_AD2 line high.
xbee.set_dio_value(IOLine.DIO2_AD2, IOValue.HIGH)
[...]
```

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

- ACK of the sent command 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 read.

#### Read digital input values

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

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

• ACK of the sent command 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), read its value (counts) with get\_adc\_value(). 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
[...]
# Get the value of the DIO 3 (analog to digital converter).
value = xbee.get_adc_value(IOLine.DIO3_AD3)
[...]
```

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

- ACK of the sent command 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:

 $examples/io/Local ADC Sample/Local ADC Sample.py\\ examples/io/Remote ADC Sample/Remote ADC Sample.py\\$ 

#### **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.

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.

To set de duty cycle value, use the method set\_pwm\_duty\_cycle() and provide the IO line configured as PWM, and the value of the duty cycle in % of the PWM. The percentage parameter 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
[...]
# Set a duty cycle of 75% to the DIO10_PWMO line (PWM output).
xbee.set_pwm_duty_cycle(IOLine.DIO10_PWMO, 75)
[...]
```

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

- ACK of the sent command 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  ${\tt XBeeException}.$

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

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

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

(continues on next page)

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

Note: In both cases (get and set), the IO line provided must be PWM capable and 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 transmit them to another node 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 IO 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
[...]
# Read an IO sample from the device.
io_sample = 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))
```

(continues on next page)

```
[...]
```

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

- ACK of the sent command 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 <code>XBeeException</code>.

## Periodic sampling

Periodic sampling allows an XBee module to take an IO sample and transmit it to another node at a periodic rate. This destination node 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 configured destination node every X seconds. A sample rate of 0s disables this feature.

#### Set the IO sampling rate

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

# Set the destination address.
xbee.set_dest_address(XBee64BitAddress.from_hex_string("0013A20040XXXXXX"))

# Set the IO sampling rate.
xbee.set_io_sampling_rate(5) # 5 seconds.
[...]
```

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

- ACK of the sent command 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
[...]
# Get the IO sampling rate.
value = xbee.get_io_sampling_rate()
[...]
```

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

- ACK of the sent command 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
[...]
# Set the destination address.
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.
xbee.set_dio_change_detection(lines)
[...]
```

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

- ACK of the sent command 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 being monitored using the  $get\_dio\_change\_detection()$  method. A None value indicates that this feature is disabled.

#### Get the DIO change detection

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

The get dio change detection () method may fail for the following reasons:

- ACK of the sent command 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 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. Then, you are notified when the local XBee 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
xbee.open()
```

(continues on next page)

```
# 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.
xbee.add_io_sample_received_callback(io_sample_callback)
[...]
```

This callback function receives three parameters when an IO sample arrives:

- The received IO sample as an IOSample object.
- The remote XBee 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 a local XBee object.
xbee = XBeeDevice("COM1", 9600)
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.
xbee.add_io_sample_received_callback(io_sample_callback)
[...]
# Unsubscribe from IO samples reception by removing the callback.
xbee.del_io_sample_received_callback(io_sample_callback)
[...]
```

The del\_io\_sample\_received\_callback() method raises 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
- Update multiple nodes

#### 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:

92

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 is performed over the serial connection. For this operation, you need the following components:

- The XBee 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 object

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

Method	Description
update_firmware(String, String, String, Int Function)	Performs a firmware update operation of the local XBee.  • xml_firmware_file (String): path of the XML file that describes the firmware to upload.  • xbee_firmware_file (String, optional): location of the XBee binary firmware file (*.gbl).  • bootloader_firmware_file (String, optional): location of the bootloader binary firmware file (*.gbl).  • timeout (Integer, optional): the maximum amount of seconds 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

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 is not API\_MODE or ESCAPED\_API\_MODE, throwing an InvalidOperatingModeException.

#### Update local XBee firmware using an XBee 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 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, String, String, String, Integer, Function)	Performs a local firmware update operation in the given target.  • target (String or "XBeeDevice"): target of the firmware upload operation. * String: serial port identifier. * "XBeeDevice": the XBee to upload its firmware.  • xml_firmware_file (String): path of the XML file that describes the firmware to upload.  • xbee_firmware_file (String, optional): location of the XBee binary firmware file (*.gbl).  • bootloader_firmware_file (String, optional): location of the bootloader binary firmware file.  • timeout (Integer, optional): the maximum amount of seconds 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

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 firmware using a serial port

### 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 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.  • xbee_firmware_file (String, optional): location
	of the XBee binary firmware file (*.ota).  • bootloader_firmware_file (String, optional): location of the XBee binary firmware file with bootloader embedded (*.otb).  • timeout (Integer, optional): the maximum amount of seconds to wait for target read oper- ations during the update process.  • progress_callback (Function, optional): func- tion to execute to receive progress information. Receives two arguments:  - The current update task as a String - The current update task percentage as an In- teger

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 FirmwareUpdateException.
- 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 a remote XBee firmware

```
[...]
XML_FIRMWARE_FILE = "/home/user/my_firmware.xml"
OTA_FIRMWARE_FILE = "/home/user/my_firmware.ota"
OTB_FIRMWARE_FILE = "/home/user/my_firmware.otb"
REMOTE_NODE_NAME = "REMOTE"
[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
# Get the network.
xnet = xbee.get_network()
# Get the remote node.
remote = xnet.discover_device(REMOTE_NODE_NAME)
# Update the remote XBee 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 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 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 a local XBee 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:

• dest\_path (String): the destination path to store

the file in.

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.  • 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 file in the specified destina-
2.6. API reference	• source_path (String): the path of the XBee files to download.

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 a profile to a local XBee
- Apply a profile to a remote XBee

To configure individual settings see Configure the XBee.

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 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 a profile to a local XBee

Applying a profile to a local XBee requires the following components:

- The local XBee 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, 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.
	• 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 a profile to a local device

```
[...]

PROFILE_PATH = "/home/user/my_profile.xpro"

[...]

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

[...]

# Apply the XBee device profile.
xbee.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 a profile to a remote XBee

Applying a profile to a remote XBee requires the following components:

• The remote XBee object instance.

• 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, 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.  • 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): 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

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 a profile to a remote device

104

```
[...]
PROFILE_PATH = "/home/user/my_profile.xpro"
REMOTE_NODE_NAME = "REMOTE"

[...]
# Instantiate a local XBee object.
xbee = XBeeDevice(...)
# Get the network.
xnet = xbee.get_network()
# Get the remote node.
```

(continues on next page)

(continued from previous page)

```
remote = xnet.discover_device(REMOTE_NODE_NAME)

[...]

# Apply the XBee 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

# 2.6.8.4 Update multiple nodes

The XBee Python Library provides a mechanism to update several nodes at once. For this, define the update tasks to perform. An update task includes:

- The node to be updated, local or remote.
- The required file(s) for the update.
- Other parameters such as the timeout or a callback to notify the progress.

There are two types of update task:

• A FwUpdateTask defines a firmware update task for a local or remote node.

```
from digi.xbee.firmware import FwUpdateTask
[...]
XML_FIRMWARE_FILE = "/home/user/my_firmware.xml"
XBEE_FIRMWARE_FILE = "/home/user/my_firmware.gbl"
BOOTLOADER_FIRMWARE_FILE = "/home/user/my_bootloader.gbl"
[...]
# Instantiate an XBee object.
xbee = XBeeDevice(...)
[...]
# Define an update progress callback for the firmware update task
def my_fw_update_cb(task_msg, percentage):
    print("%s: %%d" %(task_msg, percentage))
# Define a firmware update task for the local node
fw_update_task = FwUpdateTask(xbee, XML_FIRMWARE_FILE,
                              fw_path=XBEE_FIRMWARE_FILE,
                              bl_fw_path=BOOTLOADER_FIRMWARE_FILE,
                              progress_cb=my_fw_update_cb)
[...]
```

• A ProfileUpdateTask defines a profile update task for a local or remote node.

You can define as many update tasks as you need. Then use the update\_nodes() method of the XBeeNetwork to perform all of them.

Method	Description
update_nodes(List)	Performs the provided update tasks. It blocks until all
	tasks finish.
	• task_list (List): List of FwUpdateTask
	or ProfileUpdateTask to perform.
	The method returns a dictionary with
	the 64-bit address of the XBee as key
	and, as value, a Tuple with the XBee
	(XBeeDevice or RemoteXBeeDevice)
	and an XBeeException if the process failed
	for that node (None if it successes)

#### Update several nodes

```
from digi.xbee.firmware import ProfileUpdateTask
[...]

ROUTER_PROFILE_PATH = "/home/user/my_routers_profile.xpro"
[...]

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

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

(continues on next page)

(continued from previous page)

```
profile_tasks = []
for node in xnet.get_devices():
    if node.get_role() != Role.ROUTER:
        continue
    profile_tasks.append(ProfileUpdateTask(remote, ROUTER_PROFILE_PATH))

update_result = xnet.update_nodes(profile_tasks)

for task in tasks:
    res = update_result.get(str(task.xbee.get_64bit_addr()), None)
    res_msg = "OK"
    if res and res[1]:
        res_msg = "ERROR: %s" % str(res[1])
        print("%s: %s ---> %s" % (task.xbee, task.profile_path, res_msg))

[...]
```

To receive the status of the update process per node, provide a callback using the add\_update\_progress\_callback() method. This callback receives three arguments:

- The XBee being updated, local or remote.
- An UpdateProgressStatus with the current status.

#### Register an update progress callback

To stop listening to update progress events, use the del\_update\_progress\_callback() method to unsubscribe the already-registered callback.

#### Deregister an update progress callback

```
[...] (continues on next page)
```

(continued from previous page)

```
def cb_update_progress(node, task_str, percentage):
        [...]

xbee.add_update_progress_callback(cb_update_progress)

[...]

# Delete the callback.
xbee.del_update_progress_callback(cb_update_progress)

[...]
```

# 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 digi.xbee.devices, digi.xbee.reader, digi.xbee. sender, digi.xbee.recovery, digi.xbee.firmware, digi.xbee.profile, and digi.xbee. models.zdo. 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 to determine if log messages will be displayed in the console, written to a file, sent through a socket, etc.
- 3. A formatter to define 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>digi.xbee.devices</code> module is <code>digi.xbee.devices</code>. You can get a module name with the special attribute \ \ name \ \.

### 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. Depending on which type of handler you use, the messages created by the logger is printed in the console, to 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 it is recommended to add a formatter to a handler, and then add the handler to the logger.

When you create a formatter, you must specify the information to print and its format. This guide shows you how to create a formatter with a simple format. 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)

```
dev_logger.addHandler(handler)
[...]
```

## 2.6.9.1 Logging level

The XBee Python Library also provides a method in the digi.xbee.util.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
from digi.xbee.util.utils import enable_logger

[...]

# 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 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.

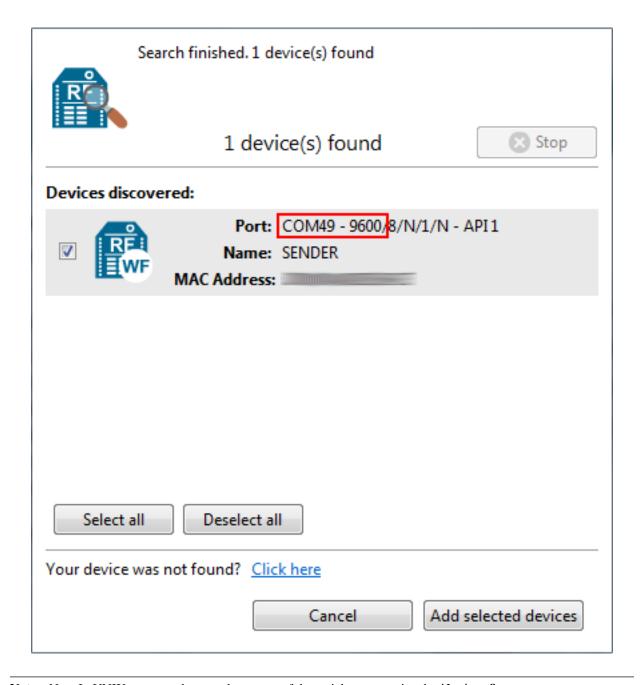
#### 2.6.11.1 What is XCTU and how do I download it?

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 v1.4.1 - 12/22/2021

- Support for new hardware variants:
  - XBee 3 Cellular LTE-M/NB-IoT (Telit)
  - XBee 3 Reduced RAM
  - S2C P5
  - XB3-DMLR
  - XB3-DMLR868
- OTA firmware update:
  - Implementation of considerations for versions 1009, 300A, 200A or prior (XBPL-375) See:
    - \* Zigbee (1009 an prior) considerations
    - \* DigiMesh (older than 300A) considerations
    - \* 802.15.4 (older than 200A) considerations
  - When updating a remote profile, let the library calculate the \*.otb file path based on the \*.xml firmware file, as it does for the \*.ota.
- · XBee Cellular:
  - Do not work with network if the XBee does not support it (XBPL-374)
  - Fix creation of IMEI when reading cellular information.
- Support to update a bunch of nodes at the same time (DAL-5285)
- Documentation:
  - Add info about the force\_settings parameter of open method (#241)
  - Add missing exportutils module to documentation.
- Set exclusive access mode to the XBee serial port (#222, #252)
- Do not stop frames reader if a serial buffer empty exception occurs (#222, #252)
- Do not use 'os.path.join()' for relative paths of zip entries (#247)

- Fix bad conditions when checking for a received packet (#242)
- Fix attribute name in find neighbors debug message (#122)
- Fix remote firmware update issue with binary file on SX devices.
- Fix protocol change issues during firmware update operation on SX devices.
- Do not reconfigure SP and SN values after a firmware update operation in P2MP protocol.
- Add new method to update salt and verifier values of Bluetooth password SRP authentication.
- Several minor bug fixes.

#### 2.6.12.2 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.

#### 2.6.12.3 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.

#### 2.6.12.4 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.

# 2.6.12.5 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.

# 2.6.12.6 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.

#### 2.6.12.7 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')
```

```
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')
          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')

Returns the code of the WiFiEncryptionType element.

code

```
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.IM = ('IM', 'IMEI')
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.NH = ('NH', 'Maximum hops')
ATStringCommand.NI = ('NI', 'Node identifier')
ATStringCommand.ND = ('ND', 'Node discover')
ATStringCommand.NJ = ('NJ', 'Join time')
ATStringCommand.NK = ('NK', 'Trust Center network key')
```

Chapter 2. Contents

132

```
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')
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')
```

2.6. API reference 133

**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:

```
SpecialByte.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

class digi.xbee.models.filesystem.FSCmdType(code, description)

# digi.xbee.models.filesystem module

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, \text{ 'Close file'})
            Read file (3) = (3, 'Read file')
            Write file (4) = (4, \text{ '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')
            Delete (47) = (47, 'Delete')
           Stat filesystem (64) = (64, 'Stat filesystem')
           Format filesystem (79) = (79, 'Format filesystem')
      code
           Returns the code of the file system command element.
               Returns Code of the file system command element.
               Return type Integer
      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
```

136

- 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

This class represents an unknown file system command.

Class constructor. Instantiates a new UnknownFSCmd object with the provided parameters.

#### **Parameters**

- raw (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()
```

This class represents a file system command request or response that includes a file or path id.

 $Bases: \ \textit{digi.xbee.models.filesystem.FSCmd}$ 

class digi.xbee.models.filesystem.FileIdCmd(cmd\_type, fid, direction=0, status=None)

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

```
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 end of the frame.

Class constructor. Instantiates a new FileIdNameCmd object with the provided parameters.

## **Parameters**

- 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 re-
            quired.
    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
```

2.6. API reference 145

classmethod create\_cmd (raw, direction=1)

Override method. Direction must be 1.

Raises

Returns OpenFileCmdResponse.

- 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.

Returns 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 exe-
              cution.
     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.

 $\textbf{Returns} \ \textit{ReadFileCmdRequest}.$ 

## 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
class digi.xbee.models.filesystem.ReadFileCmdResponse(status,
                                                                                 fid=None,
                                                                                              off-
                                                                       set=None, data=None)
     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. 0xFFFFFFFF 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.

 $\textbf{Returns} \ \textit{WriteFileCmdRequest}.$ 

## 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:

```
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
```

FileIdCmd.create cmd()

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.

 $\textbf{Returns} \ \textit{WriteFileCmdResponse}.$ 

#### 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.
```

2.6. API reference 159

**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.
         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 exe-
              cution.
     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

class digi.xbee.models.filesystem.OpenDirCmdResponse(status,

did=None,

fs\_entries=None)

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

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 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.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 <code>DirResponseFlag.ENTRY\_IS\_LAST</code> 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
            • InvalidPacketException – If the command type is not FSCmdType or direction
    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.ReadDirCmdRequest(did)

 $Bases: \verb|digi.xbee.models.filesystem.FileIdCmd| \\$ 

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 (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.ReadDirCmdResponse(status,
                                                                                        did=None,
                                                                     fs_entries=None)
     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.

**Returns** List of :class: .'FileSystemElement' inside the 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.
              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 (/).
```

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:

• '/' as path separator

• '..' to refer to the parent directory

• '.' to refer to the current path directory

- 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()
     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.GetPathIdCmdResponse(status,
                                                                                    path_id=None,
                                                                        full_path=None)
     Bases: digi.xbee.models.filesystem.FileIdCmd
```

176

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**

- status (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.

 $\textbf{Returns} \ \textit{GetPathIdCmdResponse}.$ 

### 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()
```

## 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.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.
```

**Parameters** 

Command response is received as a RenameCmdResponse.

• path\_id (Integer) - Directory path id. Set to 0x0000 for the root directory (/).

Class constructor. Instantiates a new RenameCmdRequest object with the provided parameters.

- 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.

**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.RenameCmdResponse(status)
     Bases: digi.xbee.models.filesystem.FSCmd
     This class represents a rename file system command response.
     Command response is received as a RenameCmdRequest.
     Class constructor. Instantiates a new RenameCmdResponse object with the provided parameters.
          Parameters status (FSCommandStatus or Integer) - Status of the file system command exe-
              cution.
     See also:
     FSCmd
```

180

```
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
    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

```
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
```

fs id

```
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 exe-
              cution.
     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()
```

184

```
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
```

2.6. API reference 185

• InvalidPacketException - If the bytearray length is less than 2. (cmd id + name

classmethod create\_cmd (raw, direction=0)

Override method. Direction must be 0.

Raises

**Returns** VolStatCmdRequest.

(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**

- **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. 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.

Returns 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:
```

2.6. API reference 189

Returns a dictionary with all information of the command fields.

FSCmd.status()

to dict()

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 VolFormat CmdResponse 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.

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=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 HardwareVersion.
     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')
          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')
Hardware Version. 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')
Hardware Version. XBP24C TH DIP = (45, 'XBP24C (TH-DIP): XBee PRO DIP')
Hardware Version. XB24C TH DIP = (46, 'XB24C (TH-DIP): XBee DIP')
HardwareVersion.XLR_BASEBOARD = (47, 'XLR Baseboard')
HardwareVersion.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')
```

2.6. API reference 193

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')
          HardwareVersion.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')
          HardwareVersion.CELLULAR_3_LTE_M_TELIT = (78, 'XBee 3 Cellular LTE-M/NB-IoT (Telit)')
          HardwareVersion.XBEE3_DM_LR = (80, 'XB3-DMLR')
          HardwareVersion.XBEE3 DM LR 868 = (81, 'XB3-DMLR868')
          HardwareVersion.XBEE3_RR = (82, 'XBee 3 Reduced RAM')
          HardwareVersion.S2C_P5 = (83, 'S2C P5')
     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')
          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.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

Bases: object

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**

- **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

```
class digi.xbee.models.mode.OperatingMode(code, description)
     Bases: enum. Enum
          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

 $\verb"class" digi.xbee.models.mode.APIOutputModeBit" ( \textit{code}, \textit{description})$ 

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>
 0000000000000000).

**Type** 64-bit address reserved for the coordinator (value

BROADCAST\_ADDRESS = <digi.xbee.models.address.XBee64BitAddress object>
 00000000000FFFF).

Type 64-bit broadcast address (value

UNKNOWN\_ADDRESS = <digi.xbee.models.address.XBee64BitAddress object>
 FFFFFFFFFFFFFF).

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

```
{\bf class} \ {\bf digi.xbee.models.message.XBeeMessage} \ ({\it data, remote\_node, timestamp, broad-cast=False}) \ broad-
```

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

 $\textbf{RemoteATCmdOptions.APPLY\_CHANGES} = 2$ 

**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)

 $Bases \colon \texttt{enum.Enum}$ 

Enumerates the different interfaces for the <code>UserDataRelayPacket</code> and <code>UserDataRelayOutputPacket</code>.

# 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

```
{\tt class} \ {\tt 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 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 FSCmd-Type.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.
              Returns 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:
           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
```

218

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 the 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 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

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:

**CellularAssociationIndicationStatus.SUCCESSFULLY\_CONNECTED** = (0, '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')

**CellularAssociationIndicationStatus.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')

**CellularAssociationIndicationStatus.PSM\_LOW\_POWER** = (44, 'The cellular component is in the PSM low-power state')

**CellularAssociationIndicationStatus.BYPASS\_MODE** = (47, 'Bypass mode active')

**CellularAssociationIndicationStatus.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

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

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

class digi.xbee.models.status.EmberBootloaderMessageType(code, description)

Bases: enum. Enum

Enumerates the different types of the Ember bootloader messages.

## Values:

EmberBootloaderMessageType.ACK = (6, 'ACK message')

**EmberBootloaderMessageType.NACK** = (21, 'NACK message')

EmberBootloaderMessageType.NO\_MAC\_ACK = (64, 'No MAC ACK message')

EmberBootloaderMessageType.QUERY = (81, 'Query message')

**EmberBootloaderMessageType.QUERY\_RESPONSE** = (82, 'Query response message')

**EmberBootloaderMessageType.UNKNOWN** = (255, 'Unknown')

### code

Returns the code of the *EmberBootloaderMessageType* element.

**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

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

 $\verb"class" digi.xbee.models.status. SocketState" ( \textit{code}, \textit{description})$ 

Bases: enum. Enum

Enumerates the different Socket states.

```
Values:
          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')
     code
          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:
          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, '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, 'Attempt to read past EOF (end of file)')

**Hardware failure** (0x57) = (87, 'Hardware failure')

Volume offline / format required (0x58) = (88, '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

class digi.xbee.models.status.NodeUpdateType(code, description)
 Bases: enum.Enum

```
This class lists the update types.
```

# Inherited properties:

name (String): Name (id) of the NodeUpdateType.value (String): Value of the NodeUpdateType.

### Values:

Firmware update = (0, 'Firmware update')

Profile update = (1, 'Profile update')

File system update = (2, 'File system update')

#### code

Returns the code of the NodeUpdateType element.

**Returns** Code of the NodeUpdateType element.

Return type Integer

#### desc

Returns the description of the NodeUpdateType element.

**Returns** Description of the NodeUpdateType element.

Return type String

class digi.xbee.models.status.UpdateProgressStatus(update\_type, task\_str, percent, finished)

Bases: object

This class represents the state of a update process.

Class constructor. Instantiates a new UpdateProgressState object.

## **Parameters**

- update\_type (NodeUpdateType) Type of update.
- task\_str (String) The current update task.
- **percent** (*Integer*) The current update task percentage.
- **finished** (Boolean) *True* if the update finished for the XBee, *False* otherwise.

## type

firmware or profile.

Returns The update type

Return type NodeUpdateType

Type Gets the update type

## task

Gets the update task.

**Returns** The current update task.

Return type String

## percent

Gets the progress percentage.

Returns The update task percentage

Return type Integer

### finished

Gets a boolean value indicating if the update process finished for an XBee.

#### Returns

True if the update process has finished for an XBee, False otherwise.

Return type Boolean

## 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.
         Return type List
```

2.6. API reference 235

**Type** Gets the descriptor capabilities (LSB - bit0- index 0, MSB - bit1 - index 1)

Type

• Bit0

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:

XBee16BitAddress

#### 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
```

Enumerates the available relationships between two nodes of the same network.

id

Returns the identifier of the NeighborRelationship.

**Returns** NeighborRelationship identifier.

Return type Integer

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

```
\verb|class| \verb|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.

   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:

# is\_broadcast()

ApiFrameType

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).
- $\bullet$  InvalidPacketException if the frame type is different than <code>ApiFrameType.RX\_SMS.</code>
- 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:
```

```
get_frame_type_value()
```

*ApiFrameType* 

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) 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()
```

# ${\tt needs\_id}()$

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.

258

# **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.
- 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
```

```
 \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: 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.
- 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

282

- 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.
- $\bullet \ {\tt InvalidOperatingModeException-if} \ \textit{operating\_mode} \ is \ not \ supported.$

#### 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: SUCCESS.
- 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.

```
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

## 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*.

```
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:

x64bit\_source\_addr

Returns the 64-bit source address.

**Returns** the 64-bit source address.

```
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()
```

```
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 295

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 0x000000000000FFF 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.

```
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

XBee64BitAddress

## x16bit\_dest\_addr

Returns the 16-bit destination address.

**Returns** the 16-bit destination address.

Return type XBee16BitAddress

See also:

XBeel6BitAddress

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

```
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:

```
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
```

# 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.

### 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).
- $\bullet$  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. DEVICE\_REQUEST.
- InvalidOperatingModeException if operating\_mode is not supported.

### 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.
```

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

# ${\tt get\_checksum}\,(\,)$

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.
```

### **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.

```
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 Special Byte.
- 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

See also:

SendDataRequestOptions

# file\_data

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).
- 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. 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.

### See also:

```
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

```
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.

get\_checksum()

### 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

### 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.

### See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
See also:
```

# 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.

XBeeAPIPacket.needs\_id()

**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 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 Special Byte.
- 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()
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 337

**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 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

Bases: digi.xbee.packets.base.XBeeAPIPacket

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.

### 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
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()
```

**Returns** the frame type of this packet.

Returns the frame type of this packet.

get\_frame\_type()

```
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

```
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
```

transmit\_options

```
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.
```

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

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.
Returns the transmit status.
Return type TransmitStatus
```

TransmitStatus

## frame\_id

Returns the frame ID of the packet.

**Returns** the frame ID of the packet.

Return type Integer

### get\_checksum()

See also:

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).
- 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

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.

```
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.

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.
```

2.6. API reference 365

**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
```

```
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:

See also:

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

```
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()
get_frame_type()
```

2.6. API reference 375

Returns the frame type of this packet.

See also:

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

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 395

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
```

396

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.

# 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

#### 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:

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.

#### 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

# status

Returns the socket close status.

**Returns** the socket close 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 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.
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.

2.6. API reference

407

```
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.

#### See also:

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

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
```

(SocketBindListenPacket).

response

This class represents a Socket Listen Response packet. Packet is built using the parameters of the constructor.

Socket

Bind/Listen

(0x46)

frame

#### See also:

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.

# 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

# 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

```
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.

get\_checksum()

# 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. 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** 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:
```

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.

428

• **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

### 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 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
```

SocketState

See also:

# digi.xbee.packets.wifi module

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.

```
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. Return type IOSample

2.6. API reference 433

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:
```

# is\_broadcast()

ApiFrameType

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.

```
XBeePacket.create_packet()
XBeeAPIPacket._check_api_packet()
needs_id()
Override method.
```

```
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
```

### 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:

```
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:

```
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).
```

2.6. API reference 451

**Returns** The number of addresses.

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.

```
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.

```
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.

Return type XBee64BitAddress
```

# XBee64BitAddress

See also:

**x16bit\_updater\_addr**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.

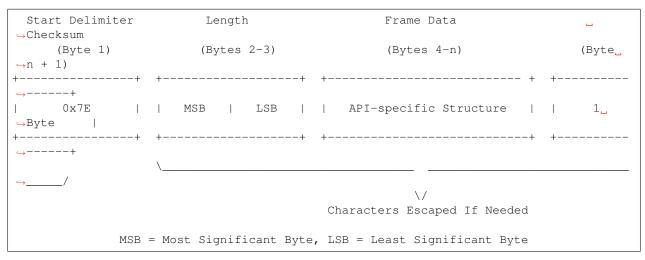
All the API and API2 logic is already included so all packet reads are independent of the XBee operating mode.

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.exportutils module

```
digi.xbee.util.exportutils.generate_network_xml (xbee, date_now=None, name=None, desc=None)

Generates the XML hierarchy representing the network of the given XBee.
```

**Params:** xbee (XBeeDevice): Local XBee node. date\_now (:class: datetime.datetime, optional, default='None'): Date

to set in the XML.

name (String, optional, default='None'): Human readable network name. desc (String, optional, default='None'): Description of the network.

Returns Generated XML hierarchy.

Return type xml.etree.ElementTree.ElementTree

## 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

```
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.
- $\bullet$   $\mbox{{\tt XModemCancelException}}$  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

# $\verb"supports_update_firmware" ()$

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.
- FirmwareUpdateException 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.
- 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.

 $\textbf{Parameters} \ \ \textbf{device} \ (\texttt{AbstractXBeeDevice}) - \textbf{XBee} \ \ \textbf{to} \ \ \textbf{get} \ \ \textbf{the} \ \ \textbf{data} \ \ \textbf{from}.$ 

Returns True if the node data has been updated, False otherwise.

Return type Boolean

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

- $\bullet$   $\mbox{\tt TimeoutException}$  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.
- 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

- $\bullet$   $\,$  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:

```
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.

**Returns** *True* if node information is complete, *False* otherwise.

Return type Boolean

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
```

See also:

• ATCommandException - If response is not as expected.

2.6. API reference 479

• XBeeException – If the XBee's communication interface is closed.

• 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.

```
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.
- $\bullet$  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:

```
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.
- XBeeException If the XBee's communication interface is closed.
- 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 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.
- $\bullet$  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.

#### 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.
- $\bullet$  ATCommandException If response is not as expected.

### update\_bluetooth\_password (new\_password, apply=True, save=True)

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.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- salt (bytes) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- **save** (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.
- ATCommandException If response is not as expected.

### **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

- $\bullet$  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.
- OperationNotSupportedException If 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

Raises FileSystemNotSupportedException - If the XBee does 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

br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

```
class digi.xbee.devices.XBeeDevice(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, exclusive=True, 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 bit-size.

- (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).
- exclusive (Boolean, optional, default=`True`) Set serial port exclusive access mode (POSIX only).
- $\bullet \ \ \textbf{comm\_iface} \ (\textit{XBeeCommunicationInterface}) \textbf{Communication interface}.$

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.

**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

<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).

- 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.

 $\verb"get_parameter" (parameter, parameter_value=None, apply=None)$ 

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.

# $\verb"add_modem_status_received_callback" (callback)$

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 <code>ExplicitDataReceived</code>.

**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.

 $\bullet \ \ The \ relay \ data \ as \ a \ \textit{UserDataRelayMessage}.$ 

### 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.

# $\verb"add_socket_state_received_callback" (callback)$

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\_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.

# ${\tt del\_modem\_status\_received\_callback}\ (callback)$

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.

# del\_fs\_frame\_received\_callback(callback)

Deletes a callback for the callback list of FileSystemFrameReceived event.

**Parameters** callback (Function) – The callback to delete.

# ${\tt get\_xbee\_device\_callbacks}\;(\,)$

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

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.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

br

Returns the BR value of the device.

Returns The BR value of the device.

**Return type** Integer

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.

Raises

```
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
```

2.6. API reference 505

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
- $\bullet$  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:

*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.
- $\bullet$  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_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.

```
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.

```
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

# 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.

```
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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### **Raises**

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

#### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.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:

    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.
- $\bullet$  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.

#### 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.
- 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.

br

Returns the BR value of the device.

Returns The BR value of the device.

Return type Integer

#### 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.
```

"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 <code>ExplicitDataReceived</code> 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.

### 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)

 $Deletes\ a\ callback\ for\ the\ callback\ list\ of\ {\it ModemStatusReceived}\ event.$ 

**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.

<sup>&</sup>quot;bitSize" -> Bit size.

<sup>&</sup>quot;stopBits" -> Stop bits.

<sup>&</sup>quot;parity" -> Parity.

```
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 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()
Returns the 16-bit address of the XBee.

Return type XBee16BitAddress
```

528

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.
- $\bullet$  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.

*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.
- $\bullet$  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

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
```

```
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.
- $\bullet$  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()
```

# 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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

# Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- 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.

### 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

 $\label{liminare_file} \begin{tabular}{ll} \textbf{update\_firmware\_file} & xbee\_firmware\_file=None, bootloader\_firmware\_file=None, timeout=None, progress\_callback=None) \end{tabular}$ 

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.
- $\bullet$  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.

- **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()
```

# 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.
- 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.

# 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.
- 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.

# 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:

ExplicitXBeeMessage
RemoteXBeeDevice

**send\_expl\_data** (remote\_xbee, data, src\_endpoint, dest\_endpoint, cluster\_id, profile\_id, trans-mit 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 *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.

# $\verb"add_user_data_relay_received_callback" (callback)$

 $Adds \ a \ callback \ for \ the \ event \ \textit{RelayDataReceived}.$ 

**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.

br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

# 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)

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 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
- apply (Boolean, optional, default=`None`) True ply 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 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:

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.
- $\bullet$  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:
```

2.6. API reference 565

*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()
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*.
- 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.

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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

```
baud rate=None,
class digi.xbee.devices.DigiPointDevice(port=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 DigiPoint XBee.
     Class constructor. Instantiates a new DigiPointDevice with the provided parameters.
          Parameters
                 • port (String) – Serial port identifier. Depends on operating system.
                   '/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()
```

### 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.
- 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:

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:

586

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.
- $\bullet$  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.

## 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.

# $\verb"add_modem_status_received_callback" (callback)$

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.

 $\verb"apply_profile" (profile_path, timeout=None, progress_callback=None)"$ 

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.

br

Returns the BR value of the device.

Returns The BR value of the device.

Return type Integer

#### 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* 

# ${\tt del\_bluetooth\_data\_received\_callback}\ (callback)$

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 <code>ExplicitDataReceived</code> 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.

# ${\tt del\_socket\_state\_received\_callback}\ (callback)$

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

592

**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 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:
     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
```

• XBeeException - If the XBee's communication interface is closed.

ullet 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.
- 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

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.
- $\bullet$  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_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.
    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()
```

```
\verb"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

- $\bullet$   $\,$  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:

```
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

# $\verb"read_device_info" (init=True, fire\_event=True)$

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.

### 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

## 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.
- $\bullet$  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.

 $\textbf{Parameters api\_output\_mode} \ (\textit{APIOutputMode}) - New \ API \ output \ mode.$ 

### **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)

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**

610

- 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()
```

# $\verb|set_pwm_duty_cycle| (io\_line, cycle)|$

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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

## **Raises**

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

## **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.

• save (Boolean, optional, default=`True`) - True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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 <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 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

618

### 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\_ontions=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:

620

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 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.

br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

#### 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)

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)

 $Deletes\ a\ callback\ for\ the\ callback\ list\ of\ \textit{SocketDataReceived}\ event.$ 

**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:

See also:

XBee64BitAddress

```
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
```

632 Chapter 2. Contents

```
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()

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.
- $\bullet$  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.
- $\bullet$  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:

```
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
```

2.6. API reference 637

• TimeoutException – If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
- $\bullet$  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\_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**

- $\bullet \ \ \textbf{remote\_xbee} \ \, (\textit{RemoteXBeeDevice}) Remote \ \, XBee \ \, that \ \, sent \ \, the \ \, data.$
- **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 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.
- $\bullet$  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.

• 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()

# $\verb"set_power_level" (power_level")$

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.
- $\bullet$  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

650

```
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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

# Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

#### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (*bytes*) *True* to apply changes, *False* otherwise, *None* to use *is\_apply\_changes\_enabled()* returned value.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
- 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.

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.
See also:
XBeeDevice
XBeeDevice.__init__()
is_device_info_complete()
     Override.
     See also:
     AbstractXBeeDevice.is_device_info_complete()
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
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.
- 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  ${\it IPDevice.}$  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>.

# $\verb|set_dest_address| (addr)$

Deprecated.

Operation not supported in this protocol. Use <code>IPDevice.set\_dest\_ip\_addr()</code> instead. This method raises an <code>AttributeError</code>.

### 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.

# $\verb"add_modem_status_received_callback" (callback)$

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.

br

Returns the BR value of the device.

Returns The BR value of the device.

Return type Integer

#### 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* 

# ${\tt del\_bluetooth\_data\_received\_callback}\ (callback)$

Deletes a callback for the callback list of BluetoothDataReceived 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.

<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\_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.
- $\bullet$  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_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.
- $\bullet$  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.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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

# $\verb"read_device_info" (init=True, fire\_event=True)$

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.

#### 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.
- $\bullet$  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\_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.
- $\bullet$  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)

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_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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

```
baud rate=None,
class digi.xbee.devices.CellularDevice(port=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:</pre>
                                                 None>, _sync_ops_timeout=4, comm_iface=None)
     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).
                 • 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.
```

2.6. API reference 681

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.

# 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.

### 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.

br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

# 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\_ip\_data\_received\_callback (callback)

Deletes a callback for the callback list of IPDataReceived event.

**Parameters** callback (Function) – The callback to delete.

# ${\tt del\_micropython\_data\_received\_callback}\ (callback)$

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:

688

<sup>&</sup>quot;parity" -> Parity.

<sup>&</sup>quot;flowControl" -> Flow control.

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.

# ${\tt del\_socket\_state\_received\_callback} \ (callback)$

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  $AbstractXBeeDevice.update\_bluetooth\_password()$ .

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  $AbstractXBeeDevice.set\_io\_configuration()$  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()

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

**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.

## 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 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 <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**

- 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()
```

# $\verb"send_data" (remote\_xbee, data, transmit\_options = 0)$

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*.
- $\bullet$  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* 

# $\verb"set_api_output_mode_value" (api_output_mode)$

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\_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()

# $\verb|set_pwm_duty_cycle| (io\_line, cycle)|$

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.

## 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, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

## **Parameters**

- $new_password(String) New Bluetooth password.$
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

#### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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 <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.

class digi.xbee.devices.LPWANDevice(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.CellularDevice

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  ${\tt AttributeError}$ .

### 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 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

# $\verb|add_micropython_data_received_callback|| (callback)$

 $Adds \ a \ callback \ for \ the \ event \ \textit{MicroPythonDataReceived}.$ 

**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

- XBeeException If the XBee's communication interface is closed.
- 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.

br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

## 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.

# ${\tt del\_fs\_frame\_received\_callback}\ (callback)$

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.

# ${\tt del\_ip\_data\_received\_callback}\ (callback)$

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:

716

#### *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  $IPDevice.get\_dest\_ip\_addr()$  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 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.
```

2.6. API reference 719

• ATCommandException - If response is not as expected.

See also:

• XBeeException – If the XBee's communication interface is closed.

• InvalidOperatingModeException – If the XBee's operating mode is not API or ESCAPED API. This method only checks the cached value of the operating mode.

```
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 <code>IPDevice.read_device_info()</code>.
          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.
```

720 Chapter 2. Contents

```
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 <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\_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.
- $\bullet$   ${\tt TimeoutException}$  If response is not received in the configured timeout.
- $\bullet$  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)

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.
- $\bullet$  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, apply=True, save=True)

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.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### **Raises**

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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

- $\bullet \ \ \hbox{\tt XBeeException-If the XBee's communication interface is closed}.$
- 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.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.

### 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.

br

Returns the BR value of the device.

Returns The BR value of the device.

Return type Integer

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.

 $\textbf{Return type} \ \textit{XBeeCommunicationInterface}$ 

### 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)

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\_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 <code>SocketDataReceived</code> 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**

- Timeout Exception 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  $IPDevice.get\_dest\_ip\_addr()$  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.
- $\bullet$  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\_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
```

2.6. API reference 747

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\_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
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.

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:

754

#### 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  ${\tt AttributeError}$ .

# 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* 

# $\verb"set_api_output_mode_value" (api_output_mode)$

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.
- $\bullet$  ATCommandException If response is not as expected.

### See also:

```
IOLine
IOMode
get_io_configuration()
```

# $\verb"set_io_sampling_rate" (\textit{rate})$

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.

 $\verb|set_parameter| (parameter, value, apply=None)|$ 

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, apply=True, save=True)

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.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If new\_password is invalid.
- 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\_bluetooth\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- 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.

### 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.
- $\bullet$  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.

```
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.

Raises 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.

# 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.

br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

### 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\_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 <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 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.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
- 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

### 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 <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**

- 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.
- $\bullet$   ${\tt TimeoutException}$  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)

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.
- $\bullet$  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* 

# $\verb|set_io_configuration| (io\_line, io\_mode)|$

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.
- $\bullet \ \, \hbox{\tt TimeoutException-If response is not received before the read timeout expires.}$
- 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.

```
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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

# Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (*bytes*) *True* to apply changes, *False* otherwise, *None* to use *is\_apply\_changes\_enabled()* returned value.

- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

## Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_file}, & xbee\_firmware\_file=None, & bootloader\_firmware\_file=None, \\ & timeout=None, & progress\_callback=None) \\ & \text{Performs a firmware update operation of the XBee.} \end{tabular}$ 

#### **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.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
```

796

### 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.

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.
- $\bullet$  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.

br

Returns the BR value of the device.

Returns The BR value of the device.

Return type Integer

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.

```
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.
- $\bullet$  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

- $\bullet \ \, \hbox{\tt TimeoutException-If response is not received before the read timeout expires.}$
- $\bullet$  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:
```

2.6. API reference 803

*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.

```
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

# 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.

```
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

- $\bullet \ \, \hbox{\tt TimeoutException-If response is not received before the read timeout expires.}$
- 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.
- 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.

*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.

```
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.
- $\bullet$  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\_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, apply=True, save=True)

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.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### Raises

- ValueError If new password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

## **Parameters**

- salt (bytes) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

```
 \begin{array}{c} \textbf{class} \ \text{digi.xbee.devices.RemoteRaw802Device} (\textit{local\_xbee}, & \textit{x64bit\_addr=None}, \\ & \textit{x16bit\_addr=None}, \textit{node\_id=None}) \\ & \text{Bases: } \textit{digi.xbee.devices.RemoteXBeeDevice} \end{array}
```

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(XBee64BitAddress) - 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.

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.

br

Returns the BR value of the device.

Returns The BR value of the device.

Return type Integer

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()
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:
    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.

```
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.
- $\bullet$  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:

*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.
- 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
          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.

### 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.
- $\bullet$  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
                • init (Boolean, optional, default=`True`) - If False only not initial-
                  ized 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

# 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.

### 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.
- $\bullet$  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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If new\_password is invalid.
- 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\_bluetooth\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
- 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.

```
class digi.xbee.devices.RemoteDigiMeshDevice(local_xbee, node_id=None)
Bases: digi.xbee.devices.RemoteXBeeDevice
x64bit_addr=None, node_id=None)
```

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()
```

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

- XBeeException If the XBee's communication interface is closed.
- 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.

#### br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

# 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.
```

**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 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

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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**

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.
- $\bullet$  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.

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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, apply=True, save=True)

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.
- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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 salt verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.

- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

 $\label{limit} \begin{tabular}{ll} \textbf{update\_firmware\_} & \textit{file}, \textit{xbee\_firmware\_} & \textit{file=None}, \textit{bootloader\_} & \textit{firmware\_} & \textit{file=None}, \\ & \textit{timeout=None}, \textit{progress\_} & \textit{callback=None}) \\ \end{tabular}$ 

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

- XBeeException If the XBee's communication interface is closed.
- 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.

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.

### See also:

```
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.

br

Returns the BR value of the device.

**Returns** The BR value of the device.

Return type Integer

#### 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()
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 <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\_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

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

```
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:
```

get\_power\_level()

Returns the power level of the XBee.

AbstractXBeeDevice.get\_parameter()

**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_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.

862

## 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
```

#### 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

864

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode 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.

*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.
- $\bullet$  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.

```
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, apply=True, save=True)

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.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If new password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

# **Parameters**

- **salt** (bytes) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is apply changes enabled() returned value.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.

- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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

- XBeeException If the XBee's communication interface is closed.
- 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.

 $\begin{array}{c} \textbf{class} \ \text{digi.xbee.devices.RemoteZigBeeDevice} \ (local\_xbee, \\ x16bit\_addr=None, node\_id=None) \\ \text{Bases: } digi.xbee.devices.RemoteXBeeDevice} \end{array}$ 

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.

#### br

Returns the BR value of the device.

Returns The BR value of the device.

Return type Integer

# 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.

```
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:
     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.

```
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\_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.
- $\bullet$  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.
- $\bullet$  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:

```
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 881

• 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**

Serial port of the local XBee associated to the remote one.

Return type XBeeSerialPort

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 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.
- 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.
- $\bullet$  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

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

```
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.

- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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, apply=True, save=True)

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.

- apply (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

- ValueError If new\_password is invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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\_salt\_verifier (salt, verifier, apply=True, save=True)

Changes the Bluetooth password of this XBee with the new one provided.

Note that your device must include Bluetooth Low Energy support.

#### **Parameters**

- **salt** (*bytes*) New Bluetooth password.
- **verifier** (bytes) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- **apply** (Boolean, optional, default=`True`) True to apply changes, False otherwise, None to use is\_apply\_changes\_enabled() returned value.
- save (Boolean, optional, default=`True`) True to save changes, False otherwise.

#### Raises

- ValueError If salt or verifier are invalid.
- TimeoutException If response is not received before the read timeout expires.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

- 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.
- 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.

## 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.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.

# 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)

### update\_nodes (task\_list)

Performs the provided update tasks. It blocks until all tasks finish.

### **Params:**

task\_list (List or tuple): List of update tasks (FwUpdateTask or ProfileUpdateTask)

## Returns

Uses the 64-bit address of the XBee as key and, as value, a Tuple with the XBee (AbstractXBeeDevice) and an XBeeException if the process failed for that node (None if it successes)

**Return type** Dictionary

# 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 RemoteXBeeDevice.

## 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()
```

# add\_update\_progress\_callback(callback)

Adds a callback for the event NetworkUpdateProgress.

**Parameters callback** (Function) – The callback. Receives three arguments. \* The XBee being updated. \* An UpdateProgressStatus with the current status.

#### See also:

```
XBeeNetwork.del_update_progress_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()
```

# ${\tt del\_end\_discovery\_scan\_callback}\ (callback)$

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\_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\_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()
```

## del\_update\_progress\_callback(callback)

Deletes a callback for the callback list of NetworkUpdateProgress.

**Parameters** callback (Function) – The callback to delete.

See also:

```
XBeeNetwork.add_update_progress_callback()
```

### get\_update\_progress\_callbacks()

Returns the list of registered callbacks for update progress. This is only for internal use.

**Returns** List of NetworkUpdateProgress events.

Return type List

#### 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.
- XBeeException If the XBee's communication interface is closed.
- InvalidOperatingModeException If the XBee's operating mode 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.

## 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:

```
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.

## add\_update\_progress\_callback(callback)

Adds a callback for the event NetworkUpdateProgress.

**Parameters callback** (Function) – The callback. Receives three arguments. \* The XBee being updated. \* An UpdateProgressStatus with the current status.

### See also:

```
XBeeNetwork.del_update_progress_callback()
```

# 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()
```

# ${\tt del\_end\_discovery\_scan\_callback}\ (callback)$

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.

 $\textbf{Parameters callback} \ (\textit{Function}) - The \ callback \ to \ delete.$ 

## 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()
```

## del\_update\_progress\_callback(callback)

Deletes a callback for the callback list of NetworkUpdateProgress.

Parameters callback (Function) - The callback to delete.

See also:

```
XBeeNetwork.add_update_progress_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.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode 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 Connect ion 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

### get\_update\_progress\_callbacks()

Returns the list of registered callbacks for update progress. This is only for internal use.

```
Returns List of NetworkUpdateProgress events.
```

**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

# 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.get_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.

## update\_nodes (task\_list)

Performs the provided update tasks. It blocks until all tasks finish.

### **Params:**

```
task_list (List or tuple): List of update tasks (FwUpdateTask or ProfileUpdateTask)
```

### Returns

Uses the 64-bit address of the XBee as key and, as value, a Tuple with the XBee (AbstractXBeeDevice) and an XBeeException if the process failed for that node (None if it successes)

**Return type** Dictionary

```
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()
```

# $\verb"add_end_discovery_scan_callback" (callback)$

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.

## add\_update\_progress\_callback(callback)

Adds a callback for the event NetworkUpdateProgress.

**Parameters callback** (Function) – The callback. Receives three arguments. \* The XBee being updated. \* An UpdateProgressStatus with the current status.

### See also:

```
XBeeNetwork.del_update_progress_callback()
```

# 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()
```

# ${\tt del\_end\_discovery\_scan\_callback}\ (callback)$

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.

 $\textbf{Parameters callback} \ (\textit{Function}) - The \ callback \ to \ delete.$ 

## 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()
```

## del\_update\_progress\_callback(callback)

Deletes a callback for the callback list of NetworkUpdateProgress.

Parameters callback (Function) - The callback to delete.

See also:

```
XBeeNetwork.add_update_progress_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.
- XBeeException If the XBee's communication interface is closed.

- InvalidOperatingModeException If the XBee's operating mode 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

### get\_update\_progress\_callbacks()

Returns the list of registered callbacks for update progress. This is only for internal use.

**Returns** List of *NetworkUpdateProgress* events.

**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

## 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.get_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.

## update\_nodes (task\_list)

Performs the provided update tasks. It blocks until all tasks finish.

### **Params:**

```
task_list (List or tuple): List of update tasks (FwUpdateTask or ProfileUpdateTask)
```

#### Returns

Uses the 64-bit address of the XBee as key and, as value, a Tuple with the XBee (AbstractXBeeDevice) and an XBeeException if the process failed for that node (None if it successes)

**Return type** Dictionary

```
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()
```

# 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.

## add\_update\_progress\_callback(callback)

Adds a callback for the event NetworkUpdateProgress.

**Parameters callback** (Function) – The callback. Receives three arguments. \* The XBee being updated. \* An UpdateProgressStatus with the current status.

#### See also:

```
XBeeNetwork.del_update_progress_callback()
```

# 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()
```

# ${\tt del\_end\_discovery\_scan\_callback}\ (callback)$

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()
```

# del\_update\_progress\_callback(callback)

Deletes a callback for the callback list of NetworkUpdateProgress.

Parameters callback (Function) - The callback to delete.

See also:

```
XBeeNetwork.add_update_progress_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 Connect ion 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

### get\_update\_progress\_callbacks()

Returns the list of registered callbacks for update progress. This is only for internal use.

**Returns** List of NetworkUpdateProgress events.

**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

# 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()
```

time\_bw\_requests=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.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.

### update\_nodes (task\_list)

Performs the provided update tasks. It blocks until all tasks finish.

### **Params:**

```
task_list (List or tuple): List of update tasks (FwUpdateTask or ProfileUpdateTask)
```

#### Returns

Uses the 64-bit address of the XBee as key and, as value, a Tuple with the XBee (AbstractXBeeDevice) and an XBeeException if the process failed for that node (None if it successes)

**Return type** Dictionary

```
\verb"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.

**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.

# add\_update\_progress\_callback(callback)

Adds a callback for the event NetworkUpdateProgress.

**Parameters callback** (Function) – The callback. Receives three arguments. \* The XBee being updated. \* An UpdateProgressStatus with the current status.

#### See also:

```
XBeeNetwork.del_update_progress_callback()
```

# 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()
```

# ${\tt del\_end\_discovery\_scan\_callback}\ (callback)$

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()
```

# del\_update\_progress\_callback(callback)

Deletes a callback for the callback list of NetworkUpdateProgress.

Parameters callback (Function) - The callback to delete.

See also:

```
XBeeNetwork.add_update_progress_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

### get\_update\_progress\_callbacks()

Returns the list of registered callbacks for update progress. This is only for internal use.

```
Returns List of NetworkUpdateProgress events.
```

**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

# 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.get_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.

### update\_nodes (task\_list)

Performs the provided update tasks. It blocks until all tasks finish.

### **Params:**

```
task_list (List or tuple): List of update tasks (FwUpdateTask or ProfileUpdateTask)
```

### Returns

Uses the 64-bit address of the XBee as key and, as value, a Tuple with the XBee (AbstractXBeeDevice) and an XBeeException if the process failed for that node (None if it successes)

Return type Dictionary

```
class digi.xbee.devices.NetworkEventType(code, description)
```

Bases: enum.Enum

Enumerates the different network event types.

# Values:

**XBee added to the network (0)** = (0, 'XBee added to the network')

**XBee removed from the network (1)** = (1, 'XBee removed from the network')

**XBee in the network updated (2) = (2, 'XBee in the network updated')** 

**Network cleared (3) = (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)

 $Bases \colon \texttt{enum.Enum}$ 

Enumerates the different network event reasons.

Values:

**Discovered XBee** (0) = (0, 'Discovered XBee')

**Discovered as XBee neighbor (1) = (1, 'Discovered as XBee neighbor')** 

**Received message from XBee (2)** = (2, 'Received message from XBee')

**Manual modification (3) = (3, 'Manual modification')** 

**Hop of a network route** (4) = (4, 'Hop of a network route')

**Read XBee information (5)** = (5, 'Read XBee information')

The firmware of the device was updated (6) = (6, 'The firmware of the device was updated')

New profile applied to the device (7) = (7, 'New profile applied to the device')

#### 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 \rightarrow 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 -> 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: 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.

### exception digi.xbee.exception.ConnectionException

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.

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.
                                                                                              API
exception digi.xbee.exception.InvalidPacketException (message='The
                                                                                      XBee
                                                                      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
     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 ex-
                                                              ecuting 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.
```

2.6. API reference 957

Exception.with\_traceback(tb) - set self.\_\_traceback\_\_ to tb and return self.

with\_traceback()

```
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.

# exception digi.xbee.exception.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()

 $Exception.with\_traceback(tb) - set \ self.\_\_traceback\_\_ \ to \ tb \ and \ return \ self.$ 

# $\textbf{exception} \ \texttt{digi.xbee.exception.} \textbf{RecoveryException}$

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.

# exception digi.xbee.filesystem.FileSystemNotSupportedException(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.
class digi.xbee.filesystem.FileProcess(f_mng, file, timeout)
     Bases: object
     This class represents a file process.
     Class constructor. Instantiates a new _FileProcess object with the provided parameters.
           Parameters
                   • (class (f_mng) - .FileSystemManager): 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()
```

# $\label{eq:write_file} \textbf{write\_file} \ (\textit{file}, \textit{offset=0}, \textit{secure=False}, \textit{options=None}, \textit{progress\_cb=None})$

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

# 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()
```

# pclose\_file (file\_id, 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** 

# $\verb|class| \verb|digi.xbee.filesystem.LocalXBeeFileSystemManager| (xbee\_device)$

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**

- **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.

Raises XBeeException – If cannot get network synchronous sleep configuration, or cannot prepare the network.

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

 $\begin{array}{lll} \textbf{class} & \texttt{digi.xbee.firmware.FwUpdateTask} \ (\textit{xbee}, & \textit{xml\_fw\_path}, & \textit{fw\_path=None}, \\ & & bl\_fw\_path=None, & timeout=None, \\ & & progress\_cb=None) \end{array}$ 

Bases: object

This class represents a firmware update process for a given XBee.

Class constructor. Instantiates a new FwUpdateTask object.

#### **Parameters**

- **xbee** (AbstractXBeeDevice) XBee to update.
- **xml\_fw\_path** (*String*) Path of the XML file that describes the firmware.
- **fw\_path** (String, optional) Location of the XBee binary firmware file.
- bl\_fw\_path (String, optional) Location of the bootloader binary firmware file
- timeout (Integer, optional) Serial port read data timeout.
- **progress\_cb** (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 ValueError – If the XBee device or the XML firmware file path are *None* or invalid. Also if the firmware binary file path or the bootloader file path are specified and does not exist.

### xbee

Gets the XBee for this task.

**Returns** The XBee to update.

Return type AbstractXBeeDevice

# xml\_path

Gets the XML firmware file path.

**Returns** The XML file path for the update task.

**Return type** String

# fw\_path

Gets the binary firmware file path.

**Returns** The binary file path for the update task.

Return type String

# bl\_path

Gets the bootloader file path.

**Returns** The bootloader file path for the update task.

Return type String

#### timeout

Gets the maximum time to wait for read operations.

**Returns** The maximum time to wait for read operations.

Return type Integer

#### callback

Returns the function to receive progress status information.

# Returns

The callback method to received progress information. *None* if not registered.

**Return type** Function

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, timeout=None, progress\_callback=None)
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
```

(continues on next page)

(continued from previous page)

```
bit 12 = AD30
bit 13 = AD40
bit 14 = AD50
bit 15 = NA0

Example: mask of 0x0C29 means DIO0, DIO3, DIO5, AD1 and AD2 enabled.

0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1
```

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 0 \times 040B means DIO0, DIO1, DIO2, DIO3 and DIO10 enabled.
0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1
```

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 = ADODIOO
bit 1 = AD1DIO1
bit 2 = AD2DIO2
bit 3 = AD3DIO3
bit 4 = AD4DIO4
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 <code>IOLine</code>, use the method <code>IOSample.has\_digital\_value()</code>.

**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 = 0Disabled SPECIAL\_FUNCTIONALITY = 1 Firmware special functionality PWM = 2PWM output ADC = 2Analog 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 0x7ff1c168ded0>)
FirmwareParity.EVEN = (1, <sphinx.ext.autodoc.importer._MockObject object at 0x7ff1c2c49310>)
FirmwareParity.ODD = (2, <sphinx.ext.autodoc.importer._MockObject object at 0x7ff1c27feed0>)
FirmwareParity.MARK = (3, <sphinx.ext.autodoc.importer._MockObject object at 0x7ff1c27fe7d0>)
FirmwareParity.SPACE = (4, <sphinx.ext.autodoc.importer._MockObject object at 0x7ff1c15246d0>)
```

#### 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 0x7ff1c222f650>) **FirmwareStopbits.SB\_2** = (1, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7ff1c1920690>) **FirmwareStopbits.SB\_1\_5** = (2, <sphinx.ext.autodoc.importer.\_MockObject object at 0x7ff1c1c04190>)

#### 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:

**name** (String): The name of this *FlashFirmwareOption*. **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.
                Returns Description of the XBeeSettingType element.
                Return type String
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

# 

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

992

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

Bases: object

This class represents a profile update process for a given XBee.

Class constructor. Instantiates a new ProfileUpdateTask object.

**Parameters** 

- **xbee** (String or 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 read operations while applying the profile.
- **progress\_cb** (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 device or the profile path are *None* or invalid.

#### xbee

Gets the XBee for this task.

**Returns** The XBee to update.

Return type AbstractXBeeDevice

#### profile path

Gets the \*.xpro file path.

**Returns** The profile path for the update task.

Return type String

#### timeout

Gets the maximum time to wait for read operations.

**Returns** The maximum time to wait for read operations.

**Return type** Integer

# callback

Returns the function to receive progress status information.

#### Returns

The callback method to received progress information. *None* if not registered.

**Return type** Function

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.

# ${\tt remove}\,(\,)$

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.
           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 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:
```

Chapter 2. Contents

1. received\_packet (XBeeAPIPacket): Received packet.

996

```
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 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
```

2. sender (RemoteXBeeDevice): Remote XBee who sent the packet.

See also:

```
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.
     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.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.XBeeDevice
                                                                               digi.xbee.devices.
                                                                       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.
          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 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.
           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 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.
```

1006

Chapter 2. Contents

```
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.
     clear()
           Remove all items from list.
     copy()
           Return a shallow copy of the list.
     count()
           Return number of occurrences of value.
           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.
     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.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.
           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
```

2.6. API reference 1009

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.

## ${\tt remove}\,(\,)$

Remove first occurrence of value.

Raises ValueError if the value is not present.

#### reverse()

Reverse IN PLACE.

#### sort()

Stable sort IN PLACE.

## ${\tt class} \ {\tt digi.xbee.reader.RouteRecordIndicatorReceived}$

```
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.
```

This event is fired when a route information packet is received.

class digi.xbee.reader.RouteInformationReceived

Bases: digi.xbee.reader.XBeeEvent

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.
- 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)

8. 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.
           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
                                                                                                  destination
                                                                                        to
                      (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.
           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.NetworkUpdateProgress
     Bases: digi.xbee.reader.XBeeEvent
     This event is fired when the progress of a running firmware update changes.
     The callbacks to handle these events will receive the following arguments:
              1. The XBee being updated.
              2. The current update task as a String.
              3. The current update task percentage as an Integer.
     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
- The remote XBee device who has sent the packet as a 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

### 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.

#### 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.

### 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.

### 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.

# ${\tt del\_route\_record\_received\_callback}\ (callback)$

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.

**Returns** List of FileSystemFrameReceived 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.

### 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

## 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).
- **exclusive** (Boolean, optional, default=`True`) Set exclusive access mode (POSIX only). A port cannot be opened in exclusive access mode if it is already open in exclusive access mode.

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.

 $\verb"apply_profile" (xbee, profile_path, timeout=None, progress\_callback=None)"$ 

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

 $\begin{tabular}{ll} \begin{tabular}{ll} \beg$ 

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.

## 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.
- $\bullet$  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.
- 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

**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 0.
- 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 0.
- 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 0.
- 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*.
- 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

2.6. API reference 1039

1040 Chapter 2. Contents

## $\mathsf{CHAPTER}\,3$

## Indices and tables

- genindex
- modindex
- search

XBee Python L	ibrary	Documentation,	Release 1.	4.1

$\sim$ L	Λ.		 $\Box$	1
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

1044 Chapter 4. License

## Python Module Index

```
d
                                           digi.xbee.recovery, 1028
                                           digi.xbee.sender, 1028
digi, 129
                                           digi.xbee.serial, 1030
digi.xbee, 129
                                           digi.xbee.util, 463
digi.xbee.comm_interface, 470
                                           digi.xbee.util.exportutils, 463
digi.xbee.devices, 473
                                           digi.xbee.util.utils,464
digi.xbee.exception, 956
                                           digi.xbee.util.xmodem, 468
digi.xbee.filesystem, 958
                                           digi.xbee.xsocket, 1034
digi.xbee.firmware, 976
digi.xbee.io,980
digi.xbee.models, 129
digi.xbee.models.accesspoint, 129
digi.xbee.models.address, 200
digi.xbee.models.atcomm, 131
digi.xbee.models.filesystem, 136
digi.xbee.models.hw, 192
digi.xbee.models.info, 195
digi.xbee.models.message, 204
digi.xbee.models.mode, 197
digi.xbee.models.options, 208
digi.xbee.models.protocol, 216
digi.xbee.models.status, 218
digi.xbee.models.zdo, 233
digi.xbee.packets, 242
digi.xbee.packets.aft, 242
digi.xbee.packets.base, 244
digi.xbee.packets.cellular, 253
digi.xbee.packets.common, 259
digi.xbee.packets.devicecloud, 305
digi.xbee.packets.digimesh, 324
digi.xbee.packets.factory,461
digi.xbee.packets.filesystem, 329
digi.xbee.packets.network, 342
digi.xbee.packets.raw, 349
digi.xbee.packets.relay, 373
digi.xbee.packets.socket, 380
digi.xbee.packets.wifi,431
digi.xbee.packets.zigbee, 443
digi.xbee.profile, 985
digi.xbee.reader, 995
```

1046 Python Module Index

A	add_data_received_callback()
AbstractXBeeDevice (class in digi.xbee.devices), 473	(digi.xbee.devices.DigiMeshDevice method), 556
accept() (digi.xbee.xsocket.socket method), 1035 AccessPoint (class in digi.xbee.models.accesspoint), 129	<pre>add_data_received_callback()       (digi.xbee.devices.DigiPointDevice method),</pre>
<pre>ack_timeout_count      (digi.xbee.packets.digimesh.RouteInformationPac</pre>	add_data_received_callback()
actual_offset (digi.xbee.models.filesystem.WriteFileContribute), 156	CmdRespon(digi.xbee.devices.LPWANDevice method), 710
ADC (digi.xbee.io.IOMode attribute), 985 add_bluetooth_data_received_callback()	<pre>add_data_received_callback()           (digi.xbee.devices.NBIoTDevice</pre>
(digi.xbee.devices.DigiMeshDevice method), 555	(digi.xbee.devices.Raw802Device method), 523
add_bluetooth_data_received_callback() (digi.xbee.devices.DigiPointDevice method), 588	<pre>add_data_received_callback()           (digi.xbee.devices.WiFiDevice method), 769 add_data_received_callback()</pre>
add_bluetooth_data_received_callback() (digi.xbee.devices.IPDevice method), 658	(digi.xbee.devices.XBeeDevice method), 498
add_bluetooth_data_received_callback()	<pre>add_data_received_callback()       (digi.xbee.devices.ZigBeeDevice method),       626</pre>
(digi.xbee.devices.NBIoTDevice method), 736 add_bluetooth_data_received_callback() (digi.xbee.devices.Raw802Device method), 522	<pre>add_data_received_callback()       (digi.xbee.reader.PacketListener method),</pre>
add_bluetooth_data_received_callback() (digi.xbee.devices.WiFiDevice method), 769	<pre>add_device_discovered_callback()      (digi.xbee.devices.DigiMeshNetwork method),</pre>
add_bluetooth_data_received_callback() (digi.xbee.devices.XBeeDevice method), 498	928 add_device_discovered_callback() (digi.xbee.devices.DigiPointNetwork method),
add_bluetooth_data_received_callback() (digi.xbee.devices.ZigBeeDevice method), 626	940 add_device_discovered_callback()
<pre>add_bluetooth_data_received_callback()           (digi.xbee.reader.PacketListener method), 1019 add_data_received_callback()</pre>	(digi.xbee.devices.Raw802Network method), 916
(digi.xbee.devices.CellularDevice method), 685	<pre>add_device_discovered_callback()      (digi.xbee.devices.XBeeNetwork method),      895</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), 1019
        940
                                                 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), 896
                                                         556
add_discovery_process_finished_callback()add_fs_frame_received_callback()
        (digi.xbee.devices.ZigBeeNetwork
                                       method),
                                                         (digi.xbee.devices.DigiPointDevice
                                                                                        method),
        904
                                                         588
add_end_discovery_scan_callback()
                                                add fs frame received callback()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                         (digi.xbee.devices.IPDevice method), 658
                                                add fs frame received callback()
add_end_discovery_scan_callback()
                                                         (digi.xbee.devices.LPWANDevice
                                                                                        method),
        (digi.xbee.devices.DigiPointNetwork method),
        941
                                                add_fs_frame_received_callback()
add_end_discovery_scan_callback()
                                                         (digi.xbee.devices.NBIoTDevice
                                                                                        method).
                                                         736
        (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), 769
        (digi.xbee.devices.ZigBeeNetwork
                                       method),
                                                add_fs_frame_received_callback()
        905
                                                         (digi.xbee.devices.XBeeDevice
                                                                                        method),
                                                         499
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),
add_expl_data_received_callback()
                                                         1020
        (digi.xbee.devices.DigiPointDevice
                                       method),
                                                add_if_not_exist()
                                                         (digi.xbee.devices.DigiMeshNetwork method),
add_expl_data_received_callback()
                                                         929
        (digi.xbee.devices.IPDevice method), 658
                                                add_if_not_exist()
                                                         (digi.xbee.devices.DigiPointNetwork method),
add_expl_data_received_callback()
                                       method),
        (digi.xbee.devices.LPWANDevice
        710
                                                add_if_not_exist()
add_expl_data_received_callback()
                                                         (digi.xbee.devices.Raw802Network
                                                                                        method),
        (digi.xbee.devices.NBIoTDevice
                                                         917
                                       method),
                                                add_if_not_exist()
add_expl_data_received_callback()
                                                         (digi.xbee.devices.XBeeNetwork
                                                                                        method),
                                       method),
                                                         902
        (digi.xbee.devices.Raw802Device
                                                add if not exist()
add_expl_data_received_callback()
                                                         (digi.xbee.devices.ZigBeeNetwork
                                                                                        method),
        (digi.xbee.devices.WiFiDevice method), 769
                                                         905
```

add\_init\_discovery\_scan\_callback() (digi.xbee.devices.NBIoTDevice method), (digi.xbee.devices.DigiMeshNetwork method), 736 add\_ip\_data\_received\_callback() add\_init\_discovery\_scan\_callback() (digi.xbee.devices.WiFiDevice method), 769 (digi.xbee.devices.DigiPointNetwork method), add ip data received callback() (digi.xbee.reader.PacketListener method), add\_init\_discovery\_scan\_callback() (digi.xbee.devices.Raw802Network method), add\_micropython\_data\_received\_callback() (digi.xbee.devices.CellularDevice method), 685 917 add\_init\_discovery\_scan\_callback() add\_micropython\_data\_received\_callback() (digi.xbee.devices.XBeeNetwork method), 896 (digi.xbee.devices.DigiMeshDevice method), add\_init\_discovery\_scan\_callback() add\_micropython\_data\_received\_callback() (digi.xbee.devices.ZigBeeNetwork method), 905 (digi.xbee.devices.DigiPointDevice add\_io\_sample\_received\_callback() (digi.xbee.devices.CellularDevice method), add\_micropython\_data\_received\_callback() (digi.xbee.devices.IPDevice method), 659 add\_io\_sample\_received\_callback() add\_micropython\_data\_received\_callback() (digi.xbee.devices.DigiMeshDevice (digi.xbee.devices.LPWANDevice method), 710 method), add\_micropython\_data\_received\_callback() add\_io\_sample\_received\_callback() (digi.xbee.devices.NBIoTDevice method), 736 (digi.xbee.devices.DigiPointDevice method), add\_micropython\_data\_received\_callback() 589 (digi.xbee.devices.Raw802Device method), 523 add io sample received callback() add micropython data received callback() (digi.xbee.devices.IPDevice method), 659 (digi.xbee.devices.WiFiDevice method), 769 add\_io\_sample\_received\_callback() add\_micropython\_data\_received\_callback() (digi.xbee.devices.LPWANDevice method), (digi.xbee.devices.XBeeDevice method), 498 add\_micropython\_data\_received\_callback() (digi.xbee.devices.ZigBeeDevice method), 627 add\_io\_sample\_received\_callback() (digi.xbee.devices.NBIoTDevice method), add\_micropython\_data\_received\_callback() (digi.xbee.reader.PacketListener method), 1019 add\_io\_sample\_received\_callback() add\_modem\_status\_received\_callback() (digi.xbee.devices.Raw802Device method), (digi.xbee.devices.CellularDevice method), 685 523 add\_modem\_status\_received\_callback() add\_io\_sample\_received\_callback() (digi.xbee.devices.DigiMeshDevice (digi.xbee.devices.WiFiDevice method), 769 add\_io\_sample\_received\_callback() add modem status received callback() (digi.xbee.devices.XBeeDevice method), (digi.xbee.devices.DigiPointDevice method), 498 589 add\_io\_sample\_received\_callback() add\_modem\_status\_received\_callback() (digi.xbee.devices.ZigBeeDevice method), (digi.xbee.devices.IPDevice method), 659 add\_modem\_status\_received\_callback() (digi.xbee.devices.LPWANDevice method), 710 add\_io\_sample\_received\_callback() (digi.xbee.reader.PacketListener method), add\_modem\_status\_received\_callback() (digi.xbee.devices.NBIoTDevice method), 737 add\_ip\_data\_received\_callback() add\_modem\_status\_received\_callback() (digi.xbee.devices.CellularDevice method), (digi.xbee.devices.Raw802Device method), 523 685 add\_modem\_status\_received\_callback() add\_ip\_data\_received\_callback() (digi.xbee.devices.WiFiDevice method), 770 (digi.xbee.devices.IPDevice method), 655 add\_modem\_status\_received\_callback() add\_ip\_data\_received\_callback() (digi.xbee.devices.XBeeDevice method), 498 (digi.xbee.devices.LPWANDevice method), add modem status received callback() 710 (digi.xbee.devices.ZigBeeDevice method), 627 add\_ip\_data\_received\_callback() add\_modem\_status\_received\_callback()

(digi.xbee.reader.PacketListener method), 1018 add_network_modified_callback()	(digi.xbee.devices.Raw802Network method), 918
(digi.xbee.devices.DigiMeshNetwork method), 930	<pre>add_packet_received_from_callback()      (digi.xbee.devices.XBeeNetwork method), 896</pre>
<pre>add_network_modified_callback()     (digi.xbee.devices.DigiPointNetwork method),     942</pre>	<pre>add_packet_received_from_callback()      (digi.xbee.devices.ZigBeeNetwork method),      906</pre>
<pre>add_network_modified_callback()     (digi.xbee.devices.Raw802Network method),     918</pre>	<pre>add_packet_received_from_callback()           (digi.xbee.reader.PacketListener method), 1018 add_remote()          (digi.xbee.devices.DigiMeshNetwork</pre>
<pre>add_network_modified_callback()       (digi.xbee.devices.XBeeNetwork</pre>	<pre>method), 930 add_remote() (digi.xbee.devices.DigiPointNetwork     method), 942</pre>
add_network_modified_callback() (digi.xbee.devices.ZigBeeNetwork method), 906	add_remote() (digi.xbee.devices.Raw802Network method), 918 add_remote() (digi.xbee.devices.XBeeNetwork
add_packet_received_callback() (digi.xbee.devices.CellularDevice method), 685	method), 903  add_remote() (digi.xbee.devices.ZigBeeNetwork method), 906
add_packet_received_callback() (digi.xbee.devices.DigiMeshDevice method), 556	add_remotes() (digi.xbee.devices.DigiMeshNetwork method), 930 add_remotes() (digi.xbee.devices.DigiPointNetwork
<pre>add_packet_received_callback()      (digi.xbee.devices.DigiPointDevice method),</pre>	<pre>method), 942 add_remotes() (digi.xbee.devices.Raw802Network</pre>
589 add_packet_received_callback() (digi.xbee.devices.IPDevice method), 659	method), 918 add_remotes() (digi.xbee.devices.XBeeNetwork method), 903
<pre>add_packet_received_callback()     (digi.xbee.devices.LPWANDevice method), 711</pre>	add_remotes() (digi.xbee.devices.ZigBeeNetwork method), 906
(digi.xbee.devices.LPWANDevice method), 711 add_packet_received_callback() (digi.xbee.devices.NBIoTDevice method), 737 add_packet_received_callback() (digi.xbee.devices.Raw802Device method), 523	<pre>method), 906 add_route_info_received_callback()         (digi.xbee.reader.PacketListener method), 1020 add_route_received_callback()         (digi.xbee.devices.CellularDevice method),         685 add_route_received_callback()         (digi.xbee.devices.DigiMeshDevice method),</pre>
(digi.xbee.devices.LPWANDevice method), 711  add_packet_received_callback() (digi.xbee.devices.NBIoTDevice method), 737  add_packet_received_callback() (digi.xbee.devices.Raw802Device method), 523  add_packet_received_callback() (digi.xbee.devices.WiFiDevice method), 770	<pre>method), 906 add_route_info_received_callback()     (digi.xbee.reader.PacketListener method), 1020 add_route_received_callback()     (digi.xbee.devices.CellularDevice method),     685 add_route_received_callback()     (digi.xbee.devices.DigiMeshDevice method),     557 add_route_received_callback()</pre>
(digi.xbee.devices.LPWANDevice method), 711 add_packet_received_callback() (digi.xbee.devices.NBIoTDevice method), 737 add_packet_received_callback() (digi.xbee.devices.Raw802Device method), 523 add_packet_received_callback()	<pre>method), 906 add_route_info_received_callback()         (digi.xbee.reader.PacketListener method), 1020 add_route_received_callback()         (digi.xbee.devices.CellularDevice method),         685 add_route_received_callback()         (digi.xbee.devices.DigiMeshDevice method),         557</pre>
(digi.xbee.devices.LPWANDevice method), 711  add_packet_received_callback() (digi.xbee.devices.NBIoTDevice method), 737  add_packet_received_callback() (digi.xbee.devices.Raw802Device method), 523  add_packet_received_callback() (digi.xbee.devices.WiFiDevice method), 770  add_packet_received_callback() (digi.xbee.devices.XBeeDevice method),  (digi.xbee.devices.XBeeDevice method),	method), 906  add_route_info_received_callback()     (digi.xbee.reader.PacketListener method), 1020  add_route_received_callback()     (digi.xbee.devices.CellularDevice method),     685  add_route_received_callback()     (digi.xbee.devices.DigiMeshDevice method),     557  add_route_received_callback()     (digi.xbee.devices.DigiPointDevice method),     589
(digi.xbee.devices.LPWANDevice method), 711  add_packet_received_callback()     (digi.xbee.devices.NBIoTDevice method), 737  add_packet_received_callback()     (digi.xbee.devices.Raw802Device method), 523  add_packet_received_callback()     (digi.xbee.devices.WiFiDevice method), 770  add_packet_received_callback()     (digi.xbee.devices.XBeeDevice method), 498  add_packet_received_callback()     (digi.xbee.devices.ZigBeeDevice method),	<pre>method), 906 add_route_info_received_callback()     (digi.xbee.reader.PacketListener method), 1020 add_route_received_callback()     (digi.xbee.devices.CellularDevice method),     685 add_route_received_callback()     (digi.xbee.devices.DigiMeshDevice method),     557 add_route_received_callback()     (digi.xbee.devices.DigiPointDevice method),     589 add_route_received_callback()     (digi.xbee.devices.IPDevice method), 659 add_route_received_callback()</pre>
(digi.xbee.devices.LPWANDevice method), 711  add_packet_received_callback()     (digi.xbee.devices.NBIoTDevice method), 737  add_packet_received_callback()     (digi.xbee.devices.Raw802Device method), 523  add_packet_received_callback()     (digi.xbee.devices.WiFiDevice method), 770  add_packet_received_callback()     (digi.xbee.devices.XBeeDevice method), 498  add_packet_received_callback()     (digi.xbee.devices.ZigBeeDevice method), 627  add_packet_received_callback()     (digi.xbee.reader.PacketListener method), 1018  add_packet_received_from_callback()     (digi.xbee.devices.DigiMeshNetwork method),	method), 906  add_route_info_received_callback()     (digi.xbee.reader.PacketListener method), 1020  add_route_received_callback()     (digi.xbee.devices.CellularDevice method),     685  add_route_received_callback()     (digi.xbee.devices.DigiMeshDevice method),     557  add_route_received_callback()     (digi.xbee.devices.DigiPointDevice method),     589  add_route_received_callback()     (digi.xbee.devices.IPDevice method), 659  add_route_received_callback()     (digi.xbee.devices.LPWANDevice method),     711  add_route_received_callback()     (digi.xbee.devices.NBIoTDevice method),     737  add_route_received_callback()
(digi.xbee.devices.LPWANDevice method), 711  add_packet_received_callback()     (digi.xbee.devices.NBIoTDevice method), 737  add_packet_received_callback()     (digi.xbee.devices.Raw802Device method), 523  add_packet_received_callback()     (digi.xbee.devices.WiFiDevice method), 770  add_packet_received_callback()     (digi.xbee.devices.XBeeDevice method), 498  add_packet_received_callback()     (digi.xbee.devices.ZigBeeDevice method), 627  add_packet_received_callback()     (digi.xbee.reader.PacketListener method), 1018  add_packet_received_from_callback()	method), 906  add_route_info_received_callback()     (digi.xbee.reader.PacketListener method), 1020  add_route_received_callback()     (digi.xbee.devices.CellularDevice method),     685  add_route_received_callback()     (digi.xbee.devices.DigiMeshDevice method),     557  add_route_received_callback()     (digi.xbee.devices.DigiPointDevice method),     589  add_route_received_callback()     (digi.xbee.devices.IPDevice method), 659  add_route_received_callback()     (digi.xbee.devices.LPWANDevice method),     711  add_route_received_callback()     (digi.xbee.devices.NBIoTDevice method),     737

```
(digi.xbee.devices.XBeeDevice
                                       method),
                                                         (digi.xbee.devices.LPWANDevice method), 711
        502
                                                add_socket_data_received_from_callback()
add route received callback()
                                                        (digi.xbee.devices.NBIoTDevice method), 737
        (digi.xbee.devices.ZigBeeDevice
                                                add_socket_data_received_from_callback()
                                       method),
                                                         (digi.xbee.devices.Raw802Device method), 524
                                                add socket data received from callback()
add route record received callback()
        (digi.xbee.reader.PacketListener method), 1020
                                                        (digi.xbee.devices.WiFiDevice method), 770
                                                add_socket_data_received_from_callback()
add sms callback()
        (digi.xbee.devices.CellularDevice
                                       method),
                                                         (digi.xbee.devices.XBeeDevice method), 499
        682
                                                add_socket_data_received_from_callback()
add_sms_callback()
                                                        (digi.xbee.devices.ZigBeeDevice method), 628
        (digi.xbee.devices.LPWANDevice
                                                add_socket_data_received_from_callback()
                                       method),
                                                         (digi.xbee.reader.PacketListener method), 1020
add_sms_callback()
                                                add_socket_state_callback()
        (digi.xbee.devices.NBIoTDevice
                                       method),
                                                         (digi.xbee.xsocket.socket method), 1038
        737
                                                add_socket_state_received_callback()
                                                         (digi.xbee.devices.CellularDevice method), 686
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), 686
                                                add_socket_state_received_callback()
add_socket_data_received_callback()
                                                        (digi.xbee.devices.DigiPointDevice
        (digi.xbee.devices.DigiMeshDevice
                                       method),
                                                        590
                                                add_socket_state_received_callback()
        557
add_socket_data_received_callback()
                                                         (digi.xbee.devices.IPDevice method), 660
        (digi.xbee.devices.DigiPointDevice
                                       method),
                                                add_socket_state_received_callback()
                                                         (digi.xbee.devices.LPWANDevice method), 711
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.IPDevice method), 659
                                                        (digi.xbee.devices.NBIoTDevice method), 737
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.LPWANDevice method), 711
                                                         (digi.xbee.devices.Raw802Device method), 524
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.NBIoTDevice method), 737
                                                         (digi.xbee.devices.WiFiDevice method), 770
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.Raw802Device method), 524
                                                         (digi.xbee.devices.XBeeDevice method), 498
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.WiFiDevice method), 770
                                                        (digi.xbee.devices.ZigBeeDevice method), 628
add_socket_data_received_callback()
                                                add_socket_state_received_callback()
        (digi.xbee.devices.XBeeDevice method), 499
                                                         (digi.xbee.reader.PacketListener method), 1019
add_socket_data_received_callback()
                                                add_update_progress_callback()
        (digi.xbee.devices.ZigBeeDevice method), 627
                                                         (digi.xbee.devices.DigiMeshNetwork method),
add_socket_data_received_callback()
        (digi.xbee.reader.PacketListener method), 1020
                                                add_update_progress_callback()
                                                        (digi.xbee.devices.DigiPointNetwork method),
add_socket_data_received_from_callback()
        (digi.xbee.devices.CellularDevice method), 686
add_socket_data_received_from_callback()add_update_progress_callback()
                                                        (digi.xbee.devices.Raw802Network
        (digi.xbee.devices.DigiMeshDevice
                                       method),
                                                                                        method),
        557
                                                         918
add_socket_data_received_from_callback()add_update_progress_callback()
        (digi.xbee.devices.DigiPointDevice
                                       method),
                                                         (digi.xbee.devices.XBeeNetwork
                                                                                        method),
add_socket_data_received_from_callback()add_update_progress_callback()
        (digi.xbee.devices.IPDevice method), 659
                                                         (digi.xbee.devices.ZigBeeNetwork
                                                                                        method),
                                                         906
add_socket_data_received_from_callback()
```

add\_user\_data\_relay\_received\_callback() method), 1013 (digi.xbee.devices.CellularDevice method), 686 append() (digi.xbee.reader.IOSampleReceived add\_user\_data\_relay\_received\_callback() method), 999 (digi.xbee.devices.DigiMeshDevice append() (digi.xbee.reader.IPDataReceived method), add\_user\_data\_relay\_received\_callback() append() (digi.xbee.reader.MicroPythonDataReceived (digi.xbee.devices.DigiPointDevice method), 1007 append() (digi.xbee.reader.ModemStatusReceived add\_user\_data\_relay\_received\_callback() method), 998 (digi.xbee.devices.IPDevice method), 660 append() (digi.xbee.reader.NetworkModified method), add\_user\_data\_relay\_received\_callback() 1000 (digi.xbee.devices.LPWANDevice method), 711 (digi.xbee.reader.NetworkUpdateProgress append() method), 1016 add\_user\_data\_relay\_received\_callback() (digi.xbee.devices.NBIoTDevice method), 738 append() (digi.xbee.reader.PacketReceived method), add\_user\_data\_relay\_received\_callback() 996 (digi.xbee.devices.Raw802Device method), 524 append() (digi.xbee.reader.PacketReceivedFrom add\_user\_data\_relay\_received\_callback() method), 997 (digi.xbee.devices.WiFiDevice method), 770 (digi.xbee.reader.RelayDataReceived append() add\_user\_data\_relay\_received\_callback() method), 1006 (digi.xbee.devices.XBeeDevice method), 498 append() (digi.xbee.reader.RouteInformationReceived add\_user\_data\_relay\_received\_callback() method), 1012 (digi.xbee.devices.ZigBeeDevice method), 628 append() (digi.xbee.reader.RouteReceived method), add\_user\_data\_relay\_received\_callback() 1013 append() (digi.xbee.reader.RouteRecordIndicatorReceived (digi.xbee.reader.PacketListener method), 1019 address (digi.xbee.models.address.XBee16BitAddress method), 1011 attribute), 201 append() (digi.xbee.reader.SMSReceived method), address (digi.xbee.models.address.XBee64BitAddress 1005 attribute), 203 append() (digi.xbee.reader.SocketDataReceived address (digi.xbee.models.address.XBeeIMEIAddress method), 1009 attribute), 203 append() (digi.xbee.reader.SocketDataReceivedFromanalog\_mask (digi.xbee.io.IOSample attribute), 983 method), 1010 analog\_values (digi.xbee.io.IOSample attribute), append() (digi.xbee.reader.SocketStateReceived method), 1008 ApiFrameType (class in digi.xbee.packets.aft), 242 append() (digi.xbee.reader.XBeeEvent method), 995 APIOutputMode (class in digi.xbee.models.mode), 197 APPEND\_DD (digi.xbee.models.options.DiscoveryOptions APIOutputModeBit (class in digi.xbee.models.mode), attribute), 213 APPEND\_RSSI (digi.xbee.models.options.DiscoveryOptions APPEND (digi.xbee.models.options.FileOpenRequestOption attribute), 213 attribute), 215 APPLY\_CHANGES (digi.xbee.models.options.RemoteATCmdOptions (digi.xbee.reader.BluetoothDataReceived attribute), 212 append() apply\_changes()(digi.xbee.devices.AbstractXBeeDevice method), 1006 (digi.xbee.reader.DataReceived method), method), 475 append() apply\_changes() (digi.xbee.devices.CellularDevice append() (digi.xbee.reader.DeviceDiscovered method), method), 686 apply\_changes()(digi.xbee.devices.DigiMeshDevice append() (digi.xbee.reader.DiscoveryProcessFinished method), 557 method), 1002 apply\_changes() (digi.xbee.devices.DigiPointDevice (digi.xbee.reader.EndDiscoveryScan append() *method*), 590 method), 1014 apply\_changes() (digi.xbee.devices.IPDevice (digi.xbee.reader.ExplicitDataReceived *method*), 660 append() method), 1003 apply\_changes() (digi.xbee.devices.LPWANDevice append() (digi.xbee.reader.FileSystemFrameReceived *method*), 711 method), 1015 apply\_changes() (digi.xbee.devices.NBIoTDevice

1052 Index

method), 738

(digi.xbee.reader.InitDiscoveryScan

append()

```
apply_changes() (digi.xbee.devices.Raw802Device apply_xbee_profile()
                                                                                                                                                    (in
                                                                                                                                                                     module
               method), 524
                                                                                                         digi.xbee.profile), 994
apply_changes() (digi.xbee.devices.RemoteDigiMeshDePte_ENCRYPTED (digi.xbee.models.options.ReceiveOptions
              method), 834
                                                                                                         attribute), 209
apply_changes() (digi.xbee.devices.RemoteDigiPointDevicei_i_to_int() (in module digi.xbee.util.utils), 466
              method), 853
                                                                                          AssociationIndicationStatus
apply changes () (digi.xbee.devices.RemoteRaw802Device
                                                                                                         digi.xbee.models.status), 223
               method), 815
                                                                                          at_command (digi.xbee.io.IOLine attribute), 980
apply_changes() (digi.xbee.devices.RemoteXBeeDevicat_response_received_cb()
              method), 797
                                                                                                         (digi.xbee.sender.PacketSender
                                                                                                                                                                   method),
apply_changes()(digi.xbee.devices.RemoteZigBeeDevice
                                                                                                         1029
               method), 874
                                                                                          ATCommand (class in digi.xbee.models.atcomm), 134
apply_changes()
                                         (digi.xbee.devices.WiFiDevice
                                                                                          ATCommandException, 956
              method), 771
                                                                                          ATCommandResponse
                                                                                                                                                   (class
                                                                                                                                                                             in
                                        (digi.xbee.devices.XBeeDevice
                                                                                                         digi.xbee.models.atcomm), 135
apply_changes()
               method), 503
                                                                                          ATCommandStatus (class in digi.xbee.models.status),
apply_changes()
                                    (digi.xbee.devices.ZigBeeDevice
                                                                                                         218
              method), 628
                                                                                          ATCommPacket (class in digi.xbee.packets.common),
apply_profile()(digi.xbee.comm_interface.XBeeCommunicationInterface
               method), 472
                                                                                          ATCommQueuePacket
                                                                                                                                                   (class
                                                                                                                                                                              in
apply_profile() (digi.xbee.devices.AbstractXBeeDevice
                                                                                                         digi.xbee.packets.common), 262
              method), 490
                                                                                          ATCommResponsePacket
                                                                                                                                                      (class
                                                                                                                                                                              in
apply_profile() (digi.xbee.devices.CellularDevice
                                                                                                         digi.xbee.packets.common), 265
               method), 687
                                                                                          ATStringCommand
                                                                                                                                                (class
                                                                                                                                                                             in
apply_profile()(digi.xbee.devices.DigiMeshDevice
                                                                                                         digi.xbee.models.atcomm), 131
              method), 558
apply_profile() (digi.xbee.devices.DigiPointDevice
              method), 590
                                                                                          baudrate
                                                                                                                 (digi.xbee.profile.FirmwareBaudrate
apply_profile()
                                            (digi.xbee.devices.IPDevice
                                                                                                         tribute), 986
              method), 660
                                                                                          bind() (digi.xbee.xsocket.socket method), 1034
apply_profile() (digi.xbee.devices.LPWANDevice
                                                                                          bl_path (digi.xbee.firmware.FwUpdateTask attribute),
               method), 712
apply_profile()
                                      (digi.xbee.devices.NBIoTDevice
                                                                                          block number (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPac
              method), 738
                                                                                                         attribute), 461
apply_profile() (digi.xbee.devices.Raw802Device
                                                                                          block size
                                                                                                                      (digi.xbee.filesystem.FileProcess
                                                                                                                                                                            at-
              method), 525
                                                                                                         tribute), 960
{\tt apply\_profile()} \ (\textit{digi.xbee.devices.RemoteDigiMeshDevices} \\ \texttt{LoothDataReceived}
                                                                                                                                                       (class
                                                                                                                                                                             in
               method), 834
                                                                                                         digi.xbee.reader), 1006
\verb|apply_profile()| (\textit{digi.xbee.devices.RemoteDigiPointDevice}_{\texttt{loader\_file}})| (\textit{digi.xbee.profile.XBeeProfile} \ \textit{at-profile()})| (\textit{digi.xbee.devices.RemoteDigiPointDevice}_{\texttt{loader\_file()}})| (\textit{digi.xbee.devices.RemoteDigiPointDevice}_{\texttt{loader\_file()}})| (\textit{digi.xbee.profile.XBeeProfile})| (\textit{digi.xbee.devices.RemoteDigiPointDevice}_{\texttt{loader\_file()}})| (\textit{digi.xbee.devices.RemoteDevice}_{\texttt{loader_file()}})| (\textit{digi.xbee.devices.RemoteDevice}_{\texttt{loader_file()}})| (\textit{digi.xbee.devices.RemoteDevices}_{\texttt{loader_file()}})| (\textit{digi.xbee.devices}_{\texttt{loader_file()}})| (\textit{digi.xbee.devices.Remo
              method), 853
                                                                                                         tribute), 993
apply_profile()(digi.xbee.devices.RemoteRaw802Devjeotloader_msg_type
               method), 815
                                                                                                         (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
apply_profile()(digi.xbee.devices.RemoteXBeeDevice
                                                                                                         attribute), 461
              method), 798
                                                                                          br (digi.xbee.devices.AbstractXBeeDevice attribute), 491
{\tt apply\_profile()} \ (\textit{digi.xbee.devices.RemoteZigBeeDevice} \ (\textit{digi.xbee.devices.CellularDevice} \ attribute), 687
               method), 874
                                                                                          br (digi.xbee.devices.DigiMeshDevice attribute), 558
apply_profile()
                                         (digi.xbee.devices.WiFiDevice
                                                                                          br (digi.xbee.devices.DigiPointDevice attribute), 590
              method), 771
                                                                                          br (digi.xbee.devices.IPDevice attribute), 660
apply_profile()
                                        (digi.xbee.devices.XBeeDevice
                                                                                          br (digi.xbee.devices.LPWANDevice attribute), 712
               method), 503
                                                                                          br (digi.xbee.devices.NBIoTDevice attribute), 738
                                     (digi.xbee.devices.ZigBeeDevice
apply_profile()
                                                                                          br (digi.xbee.devices.Raw802Device attribute), 525
              method), 628
                                                                                          br (digi.xbee.devices.RemoteDigiMeshDevice attribute),
apply_profile()
                                     (digi.xbee.serial.XBeeSerialPort
               method), 1032
```

```
br (digi.xbee.devices.RemoteDigiPointDevice attribute), callback
                                                                    (digi.xbee.profile.ProfileUpdateTask
                                                               tribute), 994
br (digi.xbee.devices.RemoteRaw802Device attribute),
                                                      CASCADE (digi.xbee.models.mode.NeighborDiscoveryMode
                                                               attribute), 199
         815
br (digi.xbee.devices.RemoteXBeeDevice attribute), 798
                                                      CellularAssociationIndicationStatus
    (digi.xbee.devices.RemoteZigBeeDevice attribute),
                                                               (class in digi.xbee.models.status), 225
                                                      CellularDevice (class in digi.xbee.devices), 680
br (digi.xbee.devices.WiFiDevice attribute), 771
                                                      change directory()
br (digi.xbee.devices.XBeeDevice attribute), 503
                                                               (digi.xbee.filesystem.LocalXBeeFileSystemManager
br (digi.xbee.devices.ZigBeeDevice attribute), 629
                                                               method), 972
BROADCAST_ADDRESS
                                                      channel (digi.xbee.models.accesspoint.AccessPoint at-
         (digi.xbee.models.address.XBee16BitAddress
                                                               tribute), 130
        attribute), 200
                                                      check_fs_support()
                                                                                        (in
                                                                                                    module
BROADCAST_ADDRESS
                                                               digi.xbee.filesystem), 975
        (digi.xbee.models.address.XBee 64Bit Address
                                                      clear() (digi.xbee.devices.DigiMeshNetwork method),
         attribute), 202
BROADCAST_PACKET (digi.xbee.models.options.ReceiveOptionsx () (digi.xbee.devices.DigiPointNetwork method),
         attribute), 209
BROADCAST_PAN (digi.xbee.models.options.TransmitOptionsear() (digi.xbee.devices.Raw802Network method),
         attribute), 210
BROADCAST_PANS_PACKET
                                                      clear() (digi.xbee.devices.XBeeNetwork method), 899
         (digi.xbee.models.options.ReceiveOptions
                                                      clear() (digi.xbee.devices.ZigBeeNetwork method),
         attribute), 209
broadcast radius (digi.xbee.packets.common.ExplicitAdderassin)gPackettligi.xbee.reader.BluetoothDataReceived
         attribute), 300
                                                               method), 1006
broadcast_radius (digi.xbee.packets.common.TransmicRaekat () (digi.xbee.reader.DataReceived method), 998
         attribute), 283
                                                      clear() (digi.xbee.reader.DeviceDiscovered method),
build_aggregate_routes()
                                                               1001
         (digi.xbee.devices. DigiMeshDevice\\
                                            method),
                                                                  (digi.xbee.reader.DiscoveryProcessFinished
                                                      clear()
                                                               method), 1002
build_frame() (in module digi.xbee.packets.factory),
                                                      clear() (digi.xbee.reader.EndDiscoveryScan method),
         463
                                                               1014
build_fs_command()
                                             module
                                                      clear()
                                                                       (digi.xbee.reader.ExplicitDataReceived
         digi.xbee.packets.filesystem), 342
                                                               method), 1003
bytearray_value(digi.xbee.profile.XBeeProfileSetting clear()
                                                                   (digi.xbee.reader.FileSystemFrameReceived
                                                               method), 1015
        attribute), 989
bytes_bad(digi.xbee.models.filesystem.VolFormatCmdResports () (digi.xbee.reader.InitDiscoveryScan method),
         attribute), 190
bytes_bad(digi.xbee.models.filesystem.VolStatCmdResponsear() (digi.xbee.reader.IOSampleReceived method),
         attribute), 187
bytes_free(digi.xbee.models.filesystem.VolFormatCmdResponse() (digi.xbee.reader.IPDataReceived method),
         attribute), 191
bytes_free(digi.xbee.models.filesystem.VolStatCmdResponser()
                                                                  (digi.xbee.reader.MicroPythonDataReceived
        attribute), 187
                                                               method), 1007
bytes_to_int() (in module digi.xbee.util.utils), 465
                                                                      (digi.xbee.reader.ModemStatusReceived
                                                      clear()
bytes_used (digi.xbee.models.filesystem.VolFormatCmdResponse method), 998
        attribute), 191
                                                      clear() (digi.xbee.reader.NetworkModified method),
bytes_used(digi.xbee.models.filesystem.VolStatCmdResponse
                                                               1000
         attribute), 187
                                                      clear()
                                                                    (digi.xbee.reader.NetworkUpdateProgress
                                                               method), 1016
C
                                                                  (digi.xbee.reader.PacketReceived method),
                                                      clear()
                                                               996
callback
              (digi.xbee.firmware.FwUpdateTask
                                                 at-
                                                                        (digi.xbee.reader.PacketReceivedFrom
                                                      clear()
        tribute), 978
                                                               method), 997
```

	0.70
clear() (digi.xbee.reader.RelayDataReceived	953
method), 1006	code (digi.xbee.devices.NetworkEventType attribute),
clear() (digi.xbee.reader.RouteInformationReceived	952
method), 1012	code (digi.xbee.io.IOValue attribute), 981
clear() (digi.xbee.reader.RouteReceived method), 1013	code (digi.xbee.models.accesspoint.WiFiEncryptionType attribute), 131
clear() (digi.xbee.reader.RouteRecordIndicatorReceived method), 1011	d code (digi.xbee.models.atcomm.SpecialByte attribute), 134
clear() (digi.xbee.reader.SMSReceived method), 1005	code (digi.xbee.models.filesystem.FSCmdType at-
clear() (digi.xbee.reader.SocketDataReceived	tribute), 136
method), 1009	code (digi.xbee.models.hw.HardwareVersion attribute),
clear() (digi.xbee.reader.SocketDataReceivedFrom	194
method), 1010	code (digi.xbee.models.hw.LegacyHardwareVersion at-
clear() (digi.xbee.reader.SocketStateReceived	tribute), 195
method), 1008	code (digi.xbee.models.mode.APIOutputMode at-
clear() (digi.xbee.reader.XBeeEvent method), 995	tribute), 198
	ewIBv4ClientPdakeri.xbee.models.mode.APIOutputModeBit
attribute), 420	attribute), 198
close() (digi.xbee.comm_interface.XBeeCommunication	
method), 470	tribute), 199
close() (digi.xbee.devices.CellularDevice method),	code (digi.xbee.models.mode.NeighborDiscoveryMode
687	attribute), 199
<pre>close() (digi.xbee.devices.DigiMeshDevice method),</pre>	code (digi.xbee.models.mode.OperatingMode attribute),
558	197
<pre>close() (digi.xbee.devices.DigiPointDevice method),</pre>	code (digi.xbee.models.options.DiscoveryOptions at-
590	tribute), 213
close() (digi.xbee.devices.IPDevice method), 660	code (digi.xbee.models.options.RegisterKeyOptions at-
close() (digi.xbee.devices.LPWANDevice method),	tribute), 214
712	code (digi.xbee.models.options.SendDataRequestOptions
close() (digi.xbee.devices.NBIoTDevice method), 738	attribute), 212
close() (digi.xbee.devices.Raw802Device method),	code (digi.xbee.models.options.SocketOption attribute),
525	214
close() (digi.xbee.devices.WiFiDevice method), 771	code (digi.xbee.models.options.XBeeLocalInterface at-
close() (digi.xbee.devices.XBeeDevice method), 493	tribute), 214
close() (digi.xbee.devices.ZigBeeDevice method), 629	code (digi.xbee.models.protocol.IPProtocol attribute),
close() (digi.xbee.profile.XBeeProfile method), 990	217
<pre>close() (digi.xbee.serial.XBeeSerialPort method),</pre>	code (digi.xbee.models.protocol.XBeeProtocol at-
1032	tribute), 217
close() (digi.xbee.xsocket.socket method), 1037	$\verb"code" (digi.x bee.models.status. Association Indication Status$
CloseDirCmdRequest (class in	attribute), 224
digi.xbee.models.filesystem), 168	code (digi.xbee.models.status.ATCommandStatus
CloseDirCmdResponse (class in	attribute), 219
digi.xbee.models.filesystem), 169	$\verb"code" (\textit{digi.xbee.models.status.} \textit{CellularAssociationIndicationStatus} \\$
CloseFileCmdRequest (class in	attribute), 225
digi.xbee.models.filesystem), 147	code (digi.xbee.models.status.DeviceCloudStatus
CloseFileCmdResponse (class in	attribute), 226
digi.xbee.models.filesystem), 148	code (digi.xbee.models.status.DiscoveryStatus at-
cluster_id(digi.xbee.models.message.ExplicitXBeeMe	
attribute), 205	$\verb"code" (digi.x bee.models.status. Ember Bootloader Message Type")$
cluster_id(digi.xbee.packets.common.ExplicitAddress	
attribute), 300	code (digi.xbee.models.status.FrameError attribute),
cluster_id(digi.xbee.packets.common.ExplicitRXIndic	
attribute), 305	code (digi.xbee.models.status.FSCommandStatus
code (digi.xbee.devices.NetworkEventReason attribute),	attribute), 231

code (digi.xbee.models.status.ModemStatus attribute),	attribute), 267
222	$\verb command  (digi.x bee.packets.common.Remote ATC ommand Packet $
code (digi.xbee.models.status.NetworkDiscoveryStatus	attribute), 275
attribute), 228	$\verb command  (digi.x bee.packets.common.Remote ATC ommand Response Packet States and States and States are also between the command Response Packet States and States are also between the command Response Packet States are also$
code (digi.xbee.models.status.NodeUpdateType at-	attribute), 278
tribute), 232	command (digi.xbee.packets.filesystem.FSRequestPacket
code (digi.xbee.models.status.PowerLevel attribute),	attribute), 330
223	command (digi.xbee.packets.filesystem.FSResponsePacket
code (digi.xbee.models.status.SocketInfoState attribute),	attribute), 333
230	command (digi.xbee.packets.filesystem.RemoteFSRequestPacket
code (digi.xbee.models.status.SocketState attribute), 230	attribute), 336
code (digi.xbee.models.status.SocketStatus attribute),	command (digi.xbee.packets.filesystem.RemoteFSResponsePacket
229	attribute), 341
code (digi.xbee.models.status.TransmitStatus attribute),	command (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket
221	attribute), 442
	Status mand (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
attribute), 227	attribute), 437
code (digi.xbee.models.status.ZigbeeRegisterStatus attribute), 228	command_value(digi.xbee.packets.common.ATCommResponsePacket attribute), 267
	command_value(digi.xbee.packets.common.RemoteATCommandRespon
code (digi.xbee.packets.aft.ApiFrameType attribute), 244	attribute), 279
code (digi.xbee.profile.FlashFirmwareOption attribute),	command_value(digi.xbee.packets.wifi.RemoteATCommandResponseWi
987	attribute), 442
comm_iface (digi.xbee.devices.CellularDevice at-	CommunicationException, 956
tribute), 687	compatibility_number
comm_iface (digi.xbee.devices.DigiMeshDevice at-	(digi.xbee.profile.XBeeProfile attribute),
tribute), 558	993
comm_iface (digi.xbee.devices.DigiPointDevice	complex_desc_supported
attribute), 591	(digi.xbee.models.zdo.NodeDescriptor at-
comm_iface (digi.xbee.devices.IPDevice attribute),	tribute), 234
661	connect () (digi.xbee.filesystem.LocalXBeeFileSystemManager
comm_iface (digi.xbee.devices.LPWANDevice at-	method), 972
tribute), 712	connect () (digi.xbee.xsocket.socket method), 1034
comm_iface (digi.xbee.devices.NBIoTDevice at-	connect_by_ap() (digi.xbee.devices.WiFiDevice
tribute), 738	method), 764
comm_iface (digi.xbee.devices.Raw802Device at-	connect_by_ssid() (digi.xbee.devices.WiFiDevice
tribute), 525	method), 764
comm_iface (digi.xbee.devices.WiFiDevice attribute),	
771	ConnectionException, 956
comm_iface (digi.xbee.devices.XBeeDevice attribute),	$\verb content_type  (\textit{digi.xbee.packets.devicecloud.SendDataRequestPacket } \\$
493	attribute), 319
comm_iface (digi.xbee.devices.ZigBeeDevice at-	COORDINATOR_ADDRESS
tribute), 629	(digi.xbee.models.address.XBee16BitAddress
command (digi.xbee.models.atcomm.ATCommand at-	attribute), 200
tribute), 135	COORDINATOR_ADDRESS
command (digi.xbee.models.atcomm.ATCommandRespons	
attribute), 135	attribute), 202
command (digi.xbee.models.atcomm.ATStringCommand	copy () (digi.xbee.reader.BluetoothDataReceived
attribute), 134	method), 1006
command (digi.xbee.packets.common.ATCommPacket	copy () (digi.xbee.reader.DataReceived method), 998
attribute), 261	сору () (digi.xbee.reader.DeviceDiscovered method), ket 1001
command (digi.xbee.packets.common.ATCommQueuePack attribute), 264	
auribule), 204 command (digi xhee packets common ATCommResponseP	· · · ·

- copy() (digi.xbee.reader.EndDiscoveryScan method), 1014
- copy() (digi.xbee.reader.ExplicitDataReceived method), 1003
- copy () (digi.xbee.reader.FileSystemFrameReceived method), 1015
- copy () (digi.xbee.reader.InitDiscoveryScan method), 1013
- copy() (digi.xbee.reader.IOSampleReceived method),
- copy() (digi.xbee.reader.IPDataReceived method), 1004
- copy () (digi.xbee.reader.MicroPythonDataReceived method), 1007
- copy() (digi.xbee.reader.ModemStatusReceived method), 998
- copy() (digi.xbee.reader.NetworkModified method), 1000
- copy () (digi.xbee.reader.NetworkUpdateProgress method), 1016
- copy () (digi.xbee.reader.PacketReceived method), 996
- copy () (digi.xbee.reader.PacketReceivedFrom method), 997
- copy () (digi.xbee.reader.RouteInformationReceived method), 1012
- copy () (digi.xbee.reader.RouteReceived method), 1013
- copy () (digi.xbee.reader.SMSReceived method), 1005
- copy () (digi.xbee.reader.SocketDataReceived method), 1009
- copy() (digi.xbee.reader.SocketDataReceivedFrom method), 1010
- copy () (digi.xbee.reader.SocketStateReceived method), 1008
- copy () (digi.xbee.reader.XBeeEvent method), 995
- count () (digi.xbee.reader.BluetoothDataReceived method), 1006
- count () (digi.xbee.reader.DataReceived method), 998
- count () (digi.xbee.reader.DeviceDiscovered method), 1001
- count () (digi.xbee.reader.DiscoveryProcessFinished method), 1002
- count () (digi.xbee.reader.ExplicitDataReceived method), 1003
- count () (digi.xbee.reader.FileSystemFrameReceived method), 1015
- count() (digi.xbee.reader.IOSampleReceived method), create\_cmd()

- 1000
- count () (digi.xbee.reader.MicroPythonDataReceived method), 1007
- count () (digi.xbee.reader.ModemStatusReceived method), 998
- count () (digi.xbee.reader.NetworkUpdateProgress method), 1016
- count() (digi.xbee.reader.PacketReceivedFrom method), 997
- count () (digi.xbee.reader.RelayDataReceived method), 1006
- count() (digi.xbee.reader.RouteReceived method).
  1013
- count () (digi.xbee.reader.SMSReceived method), 1005
- count () (digi.xbee.reader.SocketDataReceived method), 1009
- count () (digi.xbee.reader.SocketStateReceived method), 1008
- count () (digi.xbee.reader.XBeeEvent method), 995
- CREATE (digi.xbee.models.options.FileOpenRequestOption attribute), 215
- ${\tt create\_cmd()} \ ({\it digi.xbee.models.filesystem.CloseDirCmdResponse} \\ class\ {\it method}), 170$
- ${\tt create\_cmd} \ (\ ) \ ({\it digi.xbee.models.filesystem.CloseFileCmdResponse} \ class\ {\it method}), \ 149$
- ${\it create\_cmd} \ () \ ({\it digi.xbee.models.filesystem.CreateDirCmdRequest} \\ class \ {\it method}), 161$

- create\_cmd() (digi.xbee.models.filesystem.FileIdNameCmd class method), 142
- create cmd() (digi.xbee.models.filesystem.FSCmd

class method), 138 static method), 257 create\_cmd() (digi.xbee.models.filesystem.GetPathIdCnatRequest\_packet() (digi.xbee.packets.common.ATCommPacket class method), 175 static method), 260 create\_cmd() (digi.xbee.models.filesystem.GetPathIdCmdRespton\_spacket() (digi.xbee.packets.common.ATCommQueuePacket class method), 177 static method), 263 create\_cmd() (digi.xbee.models.filesystem.HashFileCmdRægaext\_packet() (digi.xbee.packets.common.ATCommResponsePacke class method), 158 static method), 266 create\_cmd() (digi.xbee.models.filesystem.HashFileCmdResponsepacket() (digi.xbee.packets.common.ExplicitAddressingPacke class method), 160 static method), 297 create\_cmd() (digi.xbee.models.filesystem.OpenDirCmdRequese\_packet() (digi.xbee.packets.common.ExplicitRXIndicatorPack class method), 164 static method), 303 create\_cmd() (digi.xbee.models.filesystem.OpenDirCmdResponse\_packet() (digi.xbee.packets.common.IODataSampleRxIndicat class method), 166 static method), 293 create\_cmd() (digi.xbee.models.filesystem.OpenFileCmdRequest\_packet() (digi.xbee.packets.common.ModemStatusPacket class method), 143 static method), 289 create\_cmd() (digi.xbee.models.filesystem.OpenFileCmdRespinesepacket() (digi.xbee.packets.common.ReceivePacket class method), 145 static method), 270 create\_cmd() (digi.xbee.models.filesystem.ReadDirCmdRequeste\_packet() (digi.xbee.packets.common.RemoteATCommandPack class method), 171 static method), 274 create\_cmd() (digi.xbee.models.filesystem.ReadDirCmdRespotuse\_packet() (digi.xbee.packets.common.RemoteATCommandResp class method), 173 static method), 278  $\verb|create_cmd()| (digi.xbee.models.filesystem.ReadFileCmdReequtst\_packet()| (digi.xbee.packets.common.TransmitPacket)| (digi.xbee.packets.common.TransmitPac$ class method), 150 static method), 282 create\_cmd() (digi.xbee.models.filesystem.ReadFileCmdResponse\_packet() (digi.xbee.packets.common.TransmitStatusPacket static method), 286 class method), 152  $\verb|create_cmd()| (\textit{digi.xbee.models.filesystem.RenameCmdRecquestPacket()} (\textit{digi.xbee.packets.devicecloud.DeviceRequestPacket()}) | (\textit{digi.xbee.models.filesystem.RenameCmdRecquestPacket()}) | (\textit{digi.xbee.packets.devicecloud.DeviceRequestPacket()}) | (\textit{digi.xbee.packets.devicecloud.DeviceCover()}) | (\textit{digi.xbee.packets.devicecloud.DeviceCover()}) | (\textit{digi.xbee.packets.devicecloud.Devi$ class method), 179 static method), 306 create\_cmd() (digi.xbee.models.filesystem.RenameCmdResponse\_packet() (digi.xbee.packets.devicecloud.DeviceResponsePack class method), 180 static method), 309 create\_cmd() (digi.xbee.models.filesystem.UnknownFSCmdate\_packet() (digi.xbee.packets.devicecloud.DeviceResponseStatu class method), 138 static method), 312 create\_cmd() (digi.xbee.models.filesystem.VolFormatCmdRequest\_packet() (digi.xbee.packets.devicecloud.FrameErrorPacket class method), 190 static method), 315 create\_cmd() (digi.xbee.models.filesystem.VolFormatCnadRespansepacket() (digi.xbee.packets.devicecloud.SendDataRequestPac class method), 192 static method), 318 create\_cmd() (digi.xbee.models.filesystem.VolStatCmdRequeste\_packet() (digi.xbee.packets.devicecloud.SendDataResponsePa class method), 185 static method), 323 create\_cmd() (digi.xbee.models.filesystem.VolStatCmdResponse\_packet() (digi.xbee.packets.digimesh.RouteInformationPacket static method), 325 class method), 187 create\_cmd() (digi.xbee.models.filesystem.WriteFileCmdRecqutest\_packet() (digi.xbee.packets.filesystem.FSRequestPacket class method), 154 static method), 329 create\_cmd() (digi.xbee.models.filesystem.WriteFileCmdResponse\_packet() (digi.xbee.packets.filesystem.FSResponse\_Packet class method), 156 static method), 332 create\_packet() (digi.xbee.packets.base.GenericXBeePackete\_packet() (digi.xbee.packets.filesystem.RemoteFSRequestPacket static method), 248 static method), 336 create\_packet() (digi.xbee.packets.base.UnknownXBeePackete\_packet() (digi.xbee.packets.filesystem.RemoteFSResponsePackete) static method), 251 static method), 340 create\_packet() (digi.xbee.packets.base.XBeeAPIPacketeate\_packet() (digi.xbee.packets.network.RXIPv4Packet static method), 247 static method), 343 create\_packet() (digi.xbee.packets.base.XBeePacket create\_packet() (digi.xbee.packets.network.TXIPv4Packet static method), 245 static method), 346 create\_packet() (digi.xbee.packets.cellular.RXSMSPacketate\_packet() (digi.xbee.packets.raw.RX16IOPacket static method), 254 static method), 371

1058 Index

create\_packet() (digi.xbee.packets.cellular.TXSMSPacketate\_packet() (digi.xbee.packets.raw.RX16Packet

```
static method), 363
                                                              static method), 454
create_packet() (digi.xbee.packets.raw.RX64IOPacketreate_packet() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatu
        static method), 366
                                                              static method), 459
create_packet() (digi.xbee.packets.raw.RX64Packet create_packet() (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
        static method), 359
                                                               static method), 447
create_packet() (digi.xbee.packets.raw.TX16Packet create_packet() (digi.xbee.packets.zigbee.RegisterJoiningDevicePack
        static method), 353
                                                              static method), 443
create_packet() (digi.xbee.packets.raw.TX64Packet create_packet() (digi.xbee.packets.zigbee.RouteRecordIndicatorPack
        static method), 350
                                                               static method), 450
create_packet() (digi.xbee.packets.raw.TXStatusPacketreate_socket_info()
        static method), 356
                                                              (digi.xbee.models.info.SocketInfo
                                                                                                     static
create_packet() (digi.xbee.packets.relay.UserDataRelayOutputPacketd), 196
        static method), 377
                                                      create_source_route()
create_packet() (digi.xbee.packets.relay.UserDataRelayPacket (digi.xbee.devices.ZigBeeDevice
                                                                                                 method),
        static method), 374
create_packet() (digi.xbee.packets.socket.SocketBindListenRacketbee_device()
        static method), 413
                                                               (digi.xbee.devices.CellularDevice
                                                                                                     class
create_packet() (digi.xbee.packets.socket.SocketClosePacket
                                                              method), 687
        static method), 400
                                                      create_xbee_device()
create_packet() (digi.xbee.packets.socket.SocketCloseResponseRdigietbee.devices.DigiMeshDevice
                                                                                                     class
        static method), 403
                                                              method), 558
create_packet() (digi.xbee.packets.socket.SocketConnectPacket_xbee_device()
                                                               (digi.xbee.devices.DigiPointDevice
        static method), 394
                                                                                                     class
create_packet() (digi.xbee.packets.socket.SocketConnectResponmePlanollot 591
        static method), 397
                                                      create xbee device()
create_packet() (digi.xbee.packets.socket.SocketCreatePacket (digi.xbee.devices.IPDevice
                                                                                                 method),
        static method), 380
create_packet() (digi.xbee.packets.socket.SocketCreateResponsePacket_device()
                                                               (digi.xbee.devices.LPWANDevice
        static method), 383
                                                                                                     class
create_packet() (digi.xbee.packets.socket.SocketListenResponseRatket), 712
        static method), 416
                                                      create_xbee_device()
create_packet() (digi.xbee.packets.socket.SocketNewIPv4Client@digitetbee.devices.NBIoTDevice class method),
        static method), 419
                                                               739
create_packet() (digi.xbee.packets.socket.SocketOptionRequiestPacket_device()
        static method), 386
                                                               (digi.xbee.devices.Raw802Device
                                                                                                     class
create_packet() (digi.xbee.packets.socket.SocketOptionResponsmentablet), 525
        static method), 390
                                                      create xbee device()
create_packet() (digi.xbee.packets.socket.SocketReceiveFromPacketi.xbee.devices.WiFiDevice class method),
        static method), 427
create_packet() (digi.xbee.packets.socket.SocketReceiveRecket_xbee_device()
                                                               (digi.xbee.devices.XBeeDevice class method),
        static method), 424
create_packet() (digi.xbee.packets.socket.SocketSendPacket
        static method), 406
                                                      create_xbee_device()
create_packet() (digi.xbee.packets.socket.SocketSendToPacket (digi.xbee.devices.ZigBeeDevice class method),
        static method), 410
create_packet() (digi.xbee.packets.socket.SocketStatePacket eDirCmdRequest
                                                                                         (class
                                                                                                        in
        static method), 430
                                                               digi.xbee.models.filesystem), 161
create_packet() (digi.xbee.packets.wifi.IODataSampleRxdadiedionWifidaeksponse
                                                                                                        in
                                                               digi.xbee.models.filesystem), 162
        static method), 432
create_packet() (digi.xbee.packets.wifi.RemoteATConumentResponsesWifiBacketPacket
                                                                                            (class
                                                                                                        in
        static method), 441
                                                               digi.xbee.packets.zigbee), 453
create_packet() (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
        static method), 436
create_packet() (digi.xbee.packets.zigbee.CreateSourceRomtePacketligi.xbee.reader.PacketListener
```

	1017	del_bluetooth_data_received_call&	back()
	(digi.xbee.models.filesystem.ReadFileCmdResponse attribute), 152	$(digi.xbee.devices. ZigBeeDevice\ methodel\_bluetooth\_data\_received\_call bluetooth\_data\_received\_call $	od), 629 back()
data	(digi.xbee.models.filesystem.WriteFileCmdRequest attribute), 154	<pre>(digi.xbee.reader.PacketListener method del_data_received_callback()</pre>	od), 1022
data	(digi.xbee.models.message.ExplicitXBeeMessage attribute), 205		method),
	(digi.xbee.models.message.IPMessage attribute), 206	<pre>del_data_received_callback()       (digi.xbee.devices.DigiMeshDevice</pre>	method),
data	(digi.xbee.models.message.SMSMessage attribute), 207	559 del_data_received_callback()	
data(	(digi.xbee.models.message.UserDataRelayMessage attribute), 208	(digi.xbee.devices.DigiPointDevice 591	method),
data	(digi.xbee.models.message.XBeeMessage attribute), 204	<pre>del_data_received_callback()           (digi.xbee.devices.IPDevice method), 6</pre>	657
data	(digi.xbee.packets.cellular.RXSMSPacket attribute), 255	<pre>del_data_received_callback()</pre>	method),
data	(digi.xbee.packets.cellular.TXSMSPacket attribute), 259	713 del_data_received_callback()	,,
data	(digi.xbee.packets.network.RXIPv4Packet attribute), 344		method),
data	(digi.xbee.packets.network.TXIPv4Packet attribute), 349	<pre>del_data_received_callback()       (digi.xbee.devices.Raw802Device</pre>	method),
data(	(digi.xbee.packets.relay.UserDataRelayOutputPacke attribute), 379	t 526 del_data_received_callback()	
data	(digi.xbee.packets.relay.UserDataRelayPacket attribute), 375	(digi.xbee.devices.WiFiDevice method) del_data_received_callback()	), 772
	Received (class in digi.xbee.reader), 997  JLT_TIME_BETWEEN_REQUESTS		method),
	(digi.xbee.devices.XBeeNetwork attribute), 892		method),
DEFAU	JLT_TIME_BETWEEN_SCANS (digi.xbee.devices.XBeeNetwork attribute),	630 del_data_received_callback()	
del k	892 Dluetooth_data_received_callback()		method),
	(digi.xbee.devices.CellularDevice method), 688	<pre>del_device_discovered_callback()</pre>	
aeı_r	<pre>cluetooth_data_received_callback()   (digi.xbee.devices.DigiMeshDevice method),</pre>	(digi.xbee.devices.DigiMeshNetwork 931	тетоа),
del_k	559 pluetooth_data_received_callback() (digi.xbee.devices.DigiPointDevice method),	<pre>del_device_discovered_callback()       (digi.xbee.devices.DigiPointNetwork       943</pre>	method),
del_k	591 Dluetooth_data_received_callback() (digi.xbee.devices.IPDevice method), 661	del_device_discovered_callback() (digi.xbee.devices.Raw802Network 919	method),
	oluetooth_data_received_callback() (digi.xbee.devices.LPWANDevice method),713	<pre>del_device_discovered_callback()           (digi.xbee.devices.XBeeNetwork           897</pre>	method),
	oluetooth_data_received_callback() (digi.xbee.devices.NBIoTDevice method), 739 oluetooth_data_received_callback()	<pre>del_device_discovered_callback()</pre>	method),
del h	(digi.xbee.devices.Raw802Device method), 526 pluetooth_data_received_callback()	907 del_discovery_process_finished_ca	allback()
	(digi.xbee.devices.WiFiDevice method), 772 pluetooth_data_received_callback()	(digi.xbee.devices.DigiMeshNetwork	
~	(digi.xbee.devices.XBeeDevice method), 500	del_discovery_process_finished_ca	allback()

```
(digi.xbee.devices.DigiPointNetwork method),
                                                         (digi.xbee.reader.PacketListener method), 1021
        943
                                                del_fs_frame_received_callback()
del_discovery_process_finished_callback()
                                                        (digi.xbee.devices.CellularDevice
                                                                                        method),
        (digi.xbee.devices.Raw802Network
                                      method),
                                                del fs frame received callback()
                                                        (digi.xbee.devices.DigiMeshDevice
                                                                                        method),
del discovery process finished callback()
        (digi.xbee.devices.XBeeNetwork method), 898
del_discovery_process_finished_callback()del_fs_frame_received_callback()
        (digi.xbee.devices.ZigBeeNetwork
                                       method),
                                                         (digi.xbee.devices.DigiPointDevice
                                                                                        method),
                                                         591
del_end_discovery_scan_callback()
                                                del_fs_frame_received_callback()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                         (digi.xbee.devices.IPDevice method), 661
                                                del_fs_frame_received_callback()
                                                        (digi.xbee.devices.LPWANDevice
del_end_discovery_scan_callback()
                                                                                        method),
        (digi.xbee.devices.DigiPointNetwork method),
                                                         713
        943
                                                del_fs_frame_received_callback()
                                                        (digi.xbee.devices.NBIoTDevice
del_end_discovery_scan_callback()
                                                                                        method),
                                                         739
        (digi.xbee.devices.Raw802Network
                                       method),
                                                del_fs_frame_received_callback()
del_end_discovery_scan_callback()
                                                         (digi.xbee.devices.Raw802Device
                                                                                        method),
        (digi.xbee.devices.XBeeNetwork
                                       method),
                                                del_fs_frame_received_callback()
del_end_discovery_scan_callback()
                                                         (digi.xbee.devices.WiFiDevice method), 772
        (digi.xbee.devices.ZigBeeNetwork
                                                del fs frame received callback()
                                       method).
        907
                                                        (digi.xbee.devices.XBeeDevice
                                                                                        method),
del_expl_data_received_callback()
        (digi.xbee.devices.CellularDevice
                                       method),
                                                del_fs_frame_received_callback()
                                                         (digi.xbee.devices.ZigBeeDevice
                                                                                        method),
del_expl_data_received_callback()
        (digi.xbee.devices.DigiMeshDevice
                                       method),
                                                del_fs_frame_received_callback()
                                                        (digi.xbee.reader.PacketListener
                                                                                        method),
del_expl_data_received_callback()
                                                         1023
        (digi.xbee.devices.DigiPointDevice
                                       method),
                                                del_init_discovery_scan_callback()
                                                         (digi.xbee.devices.DigiMeshNetwork method),
del_expl_data_received_callback()
        (digi.xbee.devices.IPDevice method), 658
                                                del_init_discovery_scan_callback()
del expl data received callback()
                                                        (digi.xbee.devices.DigiPointNetwork method),
        (digi.xbee.devices.LPWANDevice
                                       method),
        713
                                                del_init_discovery_scan_callback()
                                                        (digi.xbee.devices.Raw802Network
                                                                                        method),
del_expl_data_received_callback()
        (digi.xbee.devices.NBIoTDevice
                                       method),
                                                del_init_discovery_scan_callback()
del_expl_data_received_callback()
                                                         (digi.xbee.devices.XBeeNetwork method), 897
        (digi.xbee.devices.Raw802Device
                                       method),
                                                del_init_discovery_scan_callback()
                                                         (digi.xbee.devices.ZigBeeNetwork
                                                                                        method),
                                                         907
del_expl_data_received_callback()
        (digi.xbee.devices.WiFiDevice method), 772
                                                del_io_sample_received_callback()
                                                        (digi.xbee.devices.CellularDevice
del_expl_data_received_callback()
                                                                                        method),
        (digi.xbee.devices.XBeeDevice
                                       method),
                                                del_io_sample_received_callback()
del_expl_data_received_callback()
                                                        (digi.xbee.devices.DigiMeshDevice
                                                                                        method),
        (digi.xbee.devices.ZigBeeDevice
                                       method),
        630
                                                del_io_sample_received_callback()
del_explicit_data_received_callback()
                                                         (digi.xbee.devices.DigiPointDevice
                                                                                        method),
```

592	(digi.xbee.devices.Raw802Device method), 526
<pre>del_io_sample_received_callback()     (digi.xbee.devices.IPDevice method), 661</pre>	<pre>del_micropython_data_received_callback()</pre>
<pre>del_io_sample_received_callback()</pre>	<pre>del_micropython_data_received_callback()</pre>
(digi.xbee.devices.LPWANDevice method),	(digi.xbee.devices.XBeeDevice method), 500
713	<pre>del_micropython_data_received_callback()</pre>
<pre>del_io_sample_received_callback()</pre>	(digi.xbee.devices.ZigBeeDevice method), 630
(digi.xbee.devices.NBIoTDevice method),	<pre>del_micropython_data_received_callback()</pre>
739	(digi.xbee.reader.PacketListener method), 1022
<pre>del_io_sample_received_callback()</pre>	<pre>del_modem_status_received_callback()</pre>
(digi.xbee.devices.Raw802Device method),	(digi.xbee.devices.CellularDevice method), 688
526	<pre>del_modem_status_received_callback()</pre>
<pre>del_io_sample_received_callback()</pre>	(digi.xbee.devices.DigiMeshDevice method),
(digi.xbee.devices.WiFiDevice method), 772	559
<pre>del_io_sample_received_callback()</pre>	<pre>del_modem_status_received_callback()</pre>
(digi.xbee.devices.XBeeDevice method), 499	(digi.xbee.devices.DigiPointDevice method), 592
<pre>del_io_sample_received_callback()</pre>	<pre>del_modem_status_received_callback()</pre>
(digi.xbee.devices.ZigBeeDevice method),	(digi.xbee.devices.IPDevice method), 662
630 del_io_sample_received_callback()	<pre>del_modem_status_received_callback()           (digi.xbee.devices.LPWANDevice method), 713</pre>
(digi.xbee.reader.PacketListener method),	<pre>del_modem_status_received_callback()</pre>
1021	(digi.xbee.devices.NBIoTDevice method), 740
<pre>del_ip_data_received_callback()</pre>	<pre>del_modem_status_received_callback()</pre>
(digi.xbee.devices.CellularDevice method),	(digi.xbee.devices.Raw802Device method), 526
688	<pre>del_modem_status_received_callback()</pre>
<pre>del_ip_data_received_callback()</pre>	(digi.xbee.devices.WiFiDevice method), 772
(digi.xbee.devices.IPDevice method), 655	<pre>del_modem_status_received_callback()</pre>
<pre>del_ip_data_received_callback()</pre>	(digi.xbee.devices.XBeeDevice method), 499
(digi.xbee.devices.LPWANDevice method),	<pre>del_modem_status_received_callback()</pre>
713	(digi.xbee.devices.ZigBeeDevice method), 630
<pre>del_ip_data_received_callback()</pre>	<pre>del_modem_status_received_callback()</pre>
(digi.xbee.devices.NBIoTDevice method),	(digi.xbee.reader.PacketListener method), 1021
739	<pre>del_network_modified_callback()</pre>
<pre>del_ip_data_received_callback()      (digi.xbee.devices.WiFiDevice method),772</pre>	(digi.xbee.devices.DigiMeshNetwork method), 932
<pre>del_ip_data_received_callback()</pre>	<pre>del_network_modified_callback()</pre>
(digi.xbee.reader.PacketListener method), 1021	(digi.xbee.devices.DigiPointNetwork method), 944
<pre>del_micropython_data_received_callback()</pre>	
(digi.xbee.devices.CellularDevice method), 688	(digi.xbee.devices.Raw802Network method),
<pre>del_micropython_data_received_callback()</pre>	
(digi.xbee.devices.DigiMeshDevice method),	del_network_modified_callback()
559	(digi.xbee.devices.XBeeNetwork method), 897
<pre>del_micropython_data_received_callback()      (digi.xbee.devices.DigiPointDevice method),</pre>	del_network_modified_callback()
(digi.xbee.devices.Digit oiniDevice meinod), 592	
del_micropython_data_received_callback()	(digi.xbee.devices.ZigBeeNetwork method), 908
(digi.xbee.devices.IPDevice method), 661	del_packet_received_callback()
del_micropython_data_received_callback()	=
(digi.xbee.devices.LPWANDevice method), 713	688
del_micropython_data_received_callback()	
(digi.xbee.devices.NBIoTDevice method), 740	(digi.xbee.devices.DigiMeshDevice method),
del_micropython_data_received_callback()	

del_packet_received_callback()		(digi.xbee.devices.IPDevice method), 662
	thod),	del_route_received_callback()
592	,,	(digi.xbee.devices.LPWANDevice method),
<pre>del_packet_received_callback()</pre>		714
(digi.xbee.devices.IPDevice method), 662		<pre>del_route_received_callback()</pre>
<pre>del_packet_received_callback()</pre>		(digi.xbee.devices.NBIoTDevice method),
	thod),	740
713		<pre>del_route_received_callback()</pre>
<pre>del_packet_received_callback()           (digi.xbee.devices.NBIoTDevice method</pre>	thod),	(digi.xbee.devices.Raw802Device method), 526
740		del_route_received_callback()
del_packet_received_callback()		(digi.xbee.devices.WiFiDevice method), 773
<del>-</del>	thod),	del_route_received_callback()
526	,,,	(digi.xbee.devices.XBeeDevice method),
<pre>del_packet_received_callback()</pre>		502
(digi.xbee.devices.WiFiDevice method), 77	73	<pre>del_route_received_callback()</pre>
<pre>del_packet_received_callback()</pre>		(digi.xbee.devices.ZigBeeDevice method),
	thod),	630
499		<pre>del_route_record_received_callback()</pre>
<pre>del_packet_received_callback()</pre>	a n	(digi.xbee.reader.PacketListener method), 1022
(digi.xbee.devices.ZigBeeDevice meth	thod),	<pre>del_sms_callback()           (digi.xbee.devices.CellularDevice method),</pre>
del_packet_received_callback()		(digi.xbee.devices.CellularDevice method), 682
<del>-</del>	thod),	del_sms_callback()
1021	nou),	(digi.xbee.devices.LPWANDevice method),
<pre>del_packet_received_from_callback()</pre>		709
(digi.xbee.devices.DigiMeshNetwork meth		<pre>del_sms_callback()</pre>
932		(digi.xbee.devices.NBIoTDevice method),
<pre>del_packet_received_from_callback()</pre>		740
(digi.xbee.devices.DigiPointNetwork metl	thod),	<pre>del_sms_received_callback()</pre>
944		(digi.xbee.reader.PacketListener method),
<pre>del_packet_received_from_callback()</pre>		1021 del_socket_data_received_callback()
920	nou),	(digi.xbee.devices.CellularDevice method), 689
<pre>del_packet_received_from_callback()</pre>		del_socket_data_received_callback()
(digi.xbee.devices.XBeeNetwork method),		(digi.xbee.devices.DigiMeshDevice method),
<pre>del_packet_received_from_callback()</pre>		560
(digi.xbee.devices.ZigBeeNetwork meth	thod),	<pre>del_socket_data_received_callback()</pre>
908		(digi.xbee.devices.DigiPointDevice method),
<pre>del_packet_received_from_callback()</pre>		592
(digi.xbee.reader.PacketListener method),	1021	<pre>del_socket_data_received_callback()</pre>
del_route_info_callback()	4h o d)	(digi.xbee.devices.IPDevice method), 662
(digi.xbee.reader.PacketListener meth 1022	thod),	<pre>del_socket_data_received_callback()      (digi.xbee.devices.LPWANDevice method), 714</pre>
del_route_received_callback()		del_socket_data_received_callback()
	thod),	(digi.xbee.devices.NBIoTDevice method), 740
688		<pre>del_socket_data_received_callback()</pre>
<pre>del_route_received_callback()</pre>		(digi.xbee.devices.Raw802Device method), 527
(digi.xbee.devices.DigiMeshDevice meth	thod),	del_socket_data_received_callback()
559		(digi.xbee.devices.WiFiDevice method), 773
<pre>del_route_received_callback()</pre>		<pre>del_socket_data_received_callback()</pre>
	thod),	(digi.xbee.devices.XBeeDevice method), 500
592		del_socket_data_received_callback()
<pre>del_route_received_callback()</pre>		(digi.xbee.devices.ZigBeeDevice method), 630

del_socket_data_received_callback()	932
(digi.xbee.reader.PacketListener method), 1022	<pre>del_update_progress_callback()</pre>
del_socket_data_received_from_callback()	
(digi.xbee.devices.CellularDevice method), 689	944
del_socket_data_received_from_callback()	)del update progress callback()
(digi.xbee.devices.DigiMeshDevice method),	(digi.xbee.devices.Raw802Network method),
560	920
del_socket_data_received_from_callback()	)del update progress callback()
(digi.xbee.devices.DigiPointDevice method),	(digi.xbee.devices.XBeeNetwork method),
592	898
del_socket_data_received_from_callback()	del update progress callback()
(digi.xbee.devices.IPDevice method), 662	(digi.xbee.devices.ZigBeeNetwork method),
del_socket_data_received_from_callback()	
(digi.xbee.devices.LPWANDevice method), 714	del_user_data_relay_received_callback()
del_socket_data_received_from_callback()	
(digi.xbee.devices.NBIoTDevice method), 740	del_user_data_relay_received_callback()
del_socket_data_received_from_callback()	
(digi.xbee.devices.Raw802Device method), 527	560
	) del_user_data_relay_received_callback()
(digi.xbee.devices.WiFiDevice method), 773	(digi.xbee.devices.DigiPointDevice method),
del_socket_data_received_from_callback()	
(digi.xbee.devices.XBeeDevice method), 500	del_user_data_relay_received_callback()
del_socket_data_received_from_callback()	
(digi.xbee.devices.ZigBeeDevice method), 630	del_user_data_relay_received_callback()
del_socket_data_received_from_callback()	
(digi.xbee.reader.PacketListener method), 1022	del_user_data_relay_received_callback()
del_socket_state_callback()	(digi.xbee.devices.NBIoTDevice method), 740
(digi.xbee.xsocket.socket method), 1038	
del_socket_state_received_callback()	<pre>del_user_data_relay_received_callback()     (digi.xbee.devices.Raw802Device method), 527</pre>
(digi.xbee.devices.CellularDevice method), 689	
del_socket_state_received_callback()	<pre>del_user_data_relay_received_callback()     (digi.xbee.devices.WiFiDevice method), 773</pre>
(digi.xbee.devices.DigiMeshDevice method), 560	del_user_data_relay_received_callback()
	(digi.xbee.devices.XBeeDevice method), 500
<pre>del_socket_state_received_callback()           (digi.xbee.devices.DigiPointDevice method),</pre>	<pre>del_user_data_relay_received_callback()           (digi.xbee.devices.ZigBeeDevice method), 630</pre>
(digi.xbee.devices.Digit oiniDevice meinod), 592	
del_socket_state_received_callback()	<pre>del_user_data_relay_received_callback()           (digi.xbee.reader.PacketListener method), 1022</pre>
(digi.xbee.devices.IPDevice method), 662 del_socket_state_received_callback()	DeleteCmdRequest (class in digi.xbee.models.filesystem), 182
(digi.xbee.devices.LPWANDevice method), 714	DeleteCmdResponse (class in
del_socket_state_received_callback()	digi.xbee.models.filesystem), 183
(digi.xbee.devices.NBIoTDevice method), 740	deprecated() (in module digi.xbee.util.utils), 468
del_socket_state_received_callback()	depth (digi.xbee.models.zdo.Neighbor attribute), 241
(digi.xbee.devices.Raw802Device method), 527	
	desc (digi.xbee.models.status.NodeUpdateType attribute), 232
<pre>del_socket_state_received_callback()           (digi.xbee.devices.WiFiDevice method), 773</pre>	desc_capabilities
del_socket_state_received_callback()	
(digi.xbee.devices.XBeeDevice method), 500	tribute), 235
del_socket_state_received_callback()	description (digi.xbee.devices.NetworkEventReason
(digi.xbee.devices.ZigBeeDevice method), 630	attribute), 953
del_socket_state_received_callback()	description (digi.xbee.devices.NetworkEventType
(digi.xbee.reader.PacketListener method), 1022	attribute), 952
del_update_progress_callback()	description (digi.xbee.io.IOLine attribute), 980
(digi.xbee.devices.DigiMeshNetwork method),	description(digi.xbee.models.accesspoint.WiFiEncryptionType

```
attribute), 131
                                                                 tribute), 223
description (digi.xbee.models.atcomm.ATStringCommandescription (digi.xbee.models.status.SocketInfoState
         attribute), 134
                                                                 attribute), 230
description (digi.xbee.models.filesystem.FSCmdType
                                                        description (digi.xbee.models.status.SocketState at-
         attribute), 136
                                                                 tribute), 230
description (digi.xbee.models.hw.HardwareVersion
                                                        description
                                                                           (digi.xbee.models.status.SocketStatus
         attribute), 194
                                                                 attribute), 229
description (digi.xbee.models.mode.APIOutputMode description (digi.xbee.models.status.TransmitStatus
         attribute), 198
                                                                 attribute), 221
{\tt description} \ (\textit{digi.xbee.models.mode.APIOutputModeBittlescription} \ (\textit{digi.xbee.models.status.WiFiAssociationIndicationStatus})
         attribute), 198
                                                                 attribute), 227
description (digi.xbee.models.mode.IPAddressingModedescription (digi.xbee.models.status.ZigbeeRegisterStatus
         attribute), 199
                                                                 attribute), 228
description (digi.xbee.models.mode.NeighborDiscoveryModeription (digi.xbee.packets.aft.ApiFrameType at-
         attribute), 199
                                                                 tribute), 244
description (digi.xbee.models.mode.OperatingMode description (digi.xbee.profile.FlashFirmwareOption
         attribute), 197
                                                                 attribute), 987
description (digi.xbee.models.options.DiscoveryOptions\text{escription} (digi.xbee.profile.XBeeProfile attribute),
         attribute), 213
description (digi.xbee.models.options.RegisterKeyOptionscription (digi.xbee.profile.XBeeSettingFormat at-
         attribute), 214
                                                                 tribute), 989
description (digi.xbee.models.options.SendDataRequestOptionisption (digi.xbee.profile.XBeeSettingType at-
         attribute), 212
                                                                 tribute), 988
description (digi.xbee.models.options.SocketOption dest address(digi.xbee.packets.network.TXIPv4Packet
         attribute), 215
                                                                 attribute), 347
description (digi.xbee.models.options.XBeeLocalInterfacest_address (digi.xbee.packets.socket.SocketConnectPacket
                                                                 attribute), 395
         attribute), 214
description (digi.xbee.models.protocol.IPProtocol dest_address(digi.xbee.packets.socket.SocketSendToPacket
                                                                 attribute), 410
         attribute), 217
                  (digi.xbee.models.protocol.Role
                                                        dest_address(digi.xbee.packets.wifi.RemoteATCommandWifiPacket
description
                                                   at-
         tribute), 218
                                                                 attribute), 437
description (digi.xbee.models.protocol.XBeeProtocol DEST_ADDRESS_BINARY
         attribute), 217
                                                                 (digi.xbee.packets.socket.SocketConnectPacket
description (digi.xbee.models.status.AssociationIndicationStatus attribute), 394
         attribute), 224
                                                        DEST ADDRESS STRING
                                                                 (digi.xbee.packets.socket.SocketConnectPacket\\
description (digi.xbee.models.status.ATCommandStatus
         attribute), 219
                                                                 attribute), 394
description (digi.xbee.models.status.CellularAssociation description (digi.xbee.models.status.CellularAssociation description)
         attribute), 225
                                                                 (digi.xbee.packets.socket.SocketConnectPacket
description (digi.xbee.models.status.DeviceCloudStatus
                                                                 attribute), 395
                                                        dest endpoint (digi.xbee.models.message.ExplicitXBeeMessage
         attribute), 226
description (digi.xbee.models.status.DiscoveryStatus
                                                                 attribute), 205
         attribute), 219
                                                        dest_endpoint (digi.xbee.packets.common.ExplicitAddressingPacket
description (digi.xbee.models.status.EmberBootloaderMessageTypteribute), 300
         attribute), 229
                                                        dest_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPacket
description (digi.xbee.models.status.FrameError at-
                                                                 attribute), 305
         tribute), 226
                                                        dest_interface(digi.xbee.packets.relay.UserDataRelayPacket
description (digi.xbee.models.status.FSCommandStatus
                                                                 attribute), 375
         attribute), 231
                                                        dest_port (digi.xbee.models.message.IPMessage at-
description (digi.xbee.models.status.ModemStatus
                                                                 tribute), 206
         attribute), 222
                                                        dest_port (digi.xbee.packets.network.RXIPv4Packet
description (digi.xbee.models.status.NetworkDiscoveryStatus
                                                                 attribute), 343
                                                        dest_port (digi.xbee.packets.network.TXIPv4Packet
         attribute), 228
description (digi.xbee.models.status.PowerLevel at-
                                                                 attribute), 348
```

```
dest port (digi.xbee.packets.socket.SocketConnectPackeDeviceDiscovered (class in digi.xbee.reader), 1001
        attribute), 395
                                                DeviceRequestPacket
                                                                                (class
                                                                                             in
dest port (digi.xbee.packets.socket.SocketSendToPacket
                                                        digi.xbee.packets.devicecloud), 305
                                                DeviceResponsePacket
        attribute), 411
                                                                                 (class
                                                                                             in
destination (digi.xbee.models.zdo.Route attribute),
                                                        digi.xbee.packets.devicecloud), 309
                                                DeviceResponseStatusPacket
                                                                                    (class
                                                                                             in
determine protocol()
                                                        digi.xbee.packets.devicecloud), 312
        (digi.xbee.devices.AbstractXBeeDevice
                                                DictKeys (class in digi.xbee.packets.base), 244
       method), 476
                                                digi (module), 129
determine_protocol()
                                                digi.xbee (module), 129
        (digi.xbee.devices.CellularDevice
                                       method),
                                                digi.xbee.comm_interface(module),470
                                                digi.xbee.devices (module), 473
determine_protocol()
                                                digi.xbee.exception (module), 956
        (digi.xbee.devices.DigiMeshDevice
                                                digi.xbee.filesystem (module), 958
                                       method),
                                                digi.xbee.firmware (module), 976
determine_protocol()
                                                digi.xbee.io (module), 980
        (digi.xbee.devices.DigiPointDevice
                                                digi.xbee.models (module), 129
                                       method),
                                                digi.xbee.models.accesspoint (module), 129
determine_protocol()
                                                digi.xbee.models.address (module), 200
        (digi.xbee.devices.IPDevice method), 662
                                                digi.xbee.models.atcomm (module), 131
determine_protocol()
                                                digi.xbee.models.filesystem (module), 136
        (digi.xbee.devices.LPWANDevice
                                       method),
                                                digi.xbee.models.hw (module), 192
        714
                                                digi.xbee.models.info(module), 195
determine_protocol()
                                                digi.xbee.models.message (module), 204
        (digi.xbee.devices.NBIoTDevice
                                                digi.xbee.models.mode (module), 197
                                       method),
        740
                                                digi.xbee.models.options (module), 208
determine_protocol()
                                                digi.xbee.models.protocol (module), 216
        (digi.xbee.devices.Raw802Device
                                                digi.xbee.models.status (module), 218
                                       method),
                                                digi.xbee.models.zdo (module), 233
determine_protocol()
                                                digi.xbee.packets (module), 242
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                digi.xbee.packets.aft (module), 242
       method), 835
                                                digi.xbee.packets.base (module), 244
                                                digi.xbee.packets.cellular(module), 253
determine_protocol()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                digi.xbee.packets.common (module), 259
        method), 853
                                                digi.xbee.packets.devicecloud
determine protocol()
        (digi.xbee.devices.RemoteRaw802Device
                                                digi.xbee.packets.digimesh (module), 324
       method), 815
                                                digi.xbee.packets.factory (module), 461
determine_protocol()
                                                digi.xbee.packets.filesystem (module), 329
        (digi.xbee.devices.RemoteXBeeDevice
                                                digi.xbee.packets.network (module), 342
       method), 798
                                                digi.xbee.packets.raw (module), 349
determine protocol()
                                                digi.xbee.packets.relay(module), 373
        (digi.xbee.devices.RemoteZigBeeDevice
                                                digi.xbee.packets.socket (module), 380
       method), 874
                                                digi.xbee.packets.wifi (module), 431
determine_protocol()
                                                digi.xbee.packets.zigbee (module), 443
        (digi.xbee.devices.WiFiDevice method), 773
                                                digi.xbee.profile (module), 985
                                                digi.xbee.reader (module), 995
determine_protocol()
        (digi.xbee.devices.XBeeDevice
                                       method),
                                                digi.xbee.recovery (module), 1028
        503
                                                digi.xbee.sender (module), 1028
                                                digi.xbee.serial (module), 1030
determine_protocol()
        (digi.xbee.devices.ZigBeeDevice
                                       method),
                                                digi.xbee.util (module), 463
                                                digi.xbee.util.exportutils (module), 463
DeviceCloudStatus
                              (class
                                             in digi.xbee.util.utils(module), 464
        digi.xbee.models.status), 225
                                                digi.xbee.util.xmodem (module), 468
```

```
digi.xbee.xsocket (module), 1034
                                                                                                attribute), 167
DIGIMESH MODE (digi.xbee.models.options.ReceiveOptionsirection (digi.xbee.models.filesystem.OpenFileCmdRequest
             attribute), 209
                                                                                                attribute), 144
\verb|DIGIMESH_MODE| (digi.xbee.models.options.TransmitOptions)| to on (digi.xbee.models.filesystem.OpenFileCmdResponse)| to one of the control of the control
             attribute), 211
                                                                                                attribute), 146
DigiMeshDevice (class in digi.xbee.devices), 549
                                                                                  direction (digi.xbee.models.filesystem.ReadDirCmdRequest
DigiMeshNetwork (class in digi.xbee.devices), 928
                                                                                                attribute), 171
DigiPointDevice (class in digi.xbee.devices), 582
                                                                                  direction (digi.xbee.models.filesystem.ReadDirCmdResponse
DigiPointNetwork (class in digi.xbee.devices), 940
                                                                                                attribute), 173
                                     (digi.xbee.io.IOSample
                                                                                  {\tt direction} \ ({\it digi.xbee.models.filesystem.ReadFileCmdRequest}
digital_hsb_mask
             tribute), 982
                                                                                                attribute), 151
DIGITAL_IN (digi.xbee.io.IOMode attribute), 985
                                                                                  direction (digi.xbee.models.filesystem.ReadFileCmdResponse
digital_lsb_mask
                                     (digi.xbee.io.IOSample
                                                                           at-
                                                                                                attribute), 153
             tribute), 983
                                                                                  direction (digi.xbee.models.filesystem.RenameCmdRequest
digital_mask (digi.xbee.io.IOSample attribute), 983
                                                                                                attribute), 179
DIGITAL_OUT_HIGH (digi.xbee.io.IOMode attribute),
                                                                                  direction (digi.xbee.models.filesystem.RenameCmdResponse
                                                                                                attribute), 181
              985
DIGITAL_OUT_LOW (digi.xbee.io.IOMode attribute),
                                                                                  direction (digi.xbee.models.filesystem.UnknownFSCmd
             985
                                                                                                attribute), 139
digital values (digi.xbee.io.IOSample attribute),
                                                                                  direction (digi.xbee.models.filesystem.VolFormatCmdRequest
                                                                                                attribute), 189
direction (digi.xbee.models.filesystem.CloseDirCmdRequestection (digi.xbee.models.filesystem.VolFormatCmdResponse
             attribute), 168
                                                                                                attribute), 191
direction (digi.xbee.models.filesystem.CloseDirCmdResponsection (digi.xbee.models.filesystem.VolStatCmdRequest
             attribute), 170
                                                                                                attribute), 186
{\tt direction} \ (\textit{digi.xbee.models.filesystem.CloseFileCmdRespiest} ection \ (\textit{digi.xbee.models.filesystem.VolStatCmdResponse})
             attribute), 147
                                                                                                attribute), 188
direction (digi.xbee.models.filesystem.CloseFileCmdRespinexection (digi.xbee.models.filesystem.WriteFileCmdRequest
             attribute), 149
                                                                                                attribute), 155
direction (digi.xbee.models.filesystem.CreateDirCmdRequestion (digi.xbee.models.filesystem.WriteFileCmdResponse
             attribute), 161
                                                                                                attribute), 156
direction (digi.xbee.models.filesystem.CreateDirCmdRepporBesponseFlag
                                                                                                                                    (class
                                                                                                                                                               in
             attribute), 163
                                                                                                digi.xbee.models.options), 215
attribute), 182
                                                                                                attribute), 211
direction (digi.xbee.models.filesystem.DeleteCmdRespoDsESABLE_ACK (digi.xbee.models.options.TransmitOptions
             attribute), 184
                                                                                                attribute), 210
direction (digi.xbee.models.filesystem.FileIdCmd at- disable_bluetooth()
              tribute), 140
                                                                                                (digi.xbee.devices.AbstractXBeeDevice
direction (digi.xbee.models.filesystem.FileIdNameCmd
                                                                                                method), 488
                                                                                  disable bluetooth()
             attribute), 142
direction
                             (digi.xbee.models.filesystem.FSCmd
                                                                                                (digi.xbee.devices.CellularDevice
                                                                                                                                                     method),
             attribute), 137
direction(digi.xbee.models.filesystem.GetPathIdCmdRequesable_bluetooth()
                                                                                                (digi.xbee.devices.DigiMeshDevice
             attribute), 175
                                                                                                                                                     method),
                                                                                                560
direction (digi.xbee.models.filesystem.GetPathIdCmdResponse
             attribute), 177
                                                                                  disable bluetooth()
direction (digi.xbee.models.filesystem.HashFileCmdRequest
                                                                                                (digi.xbee.devices.DigiPointDevice
                                                                                                                                                     method),
             attribute), 158
direction (digi.xbee.models.filesystem.HashFileCmdResportsable_bluetooth() (digi.xbee.devices.IPDevice
             attribute), 160
                                                                                                method), 662
direction (digi.xbee.models.filesystem.OpenDirCmdRequiessable bluetooth()
             attribute), 165
                                                                                                (digi.xbee.devices.LPWANDevice
                                                                                                                                                     method),
direction (digi.xbee.models.filesystem.OpenDirCmdResponse
                                                                                                714
```

```
disable bluetooth()
                                                    discover devices()
        (digi.xbee.devices.NBIoTDevice
                                          method).
                                                            (digi.xbee.devices.DigiMeshNetwork method),
                                                            932
        741
disable_bluetooth()
                                                   discover_devices()
        (digi.xbee.devices.Raw802Device
                                          method),
                                                            (digi.xbee.devices.DigiPointNetwork method),
                                                   discover devices()
disable bluetooth()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                            (digi.xbee.devices.Raw802Network
                                                                                              method),
        method), 835
                                                            920
disable_bluetooth()
                                                    discover_devices()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                            (digi.xbee.devices.XBeeNetwork
                                                                                              method),
        method), 854
disable_bluetooth()
                                                   discover_devices()
        (digi.xbee.devices.RemoteRaw802Device
                                                            (digi.xbee.devices.ZigBeeNetwork
                                                                                              method),
        method), 816
disable_bluetooth()
                                                   DISCOVER_MYSELF (digi.xbee.models.options.DiscoveryOptions
        (digi.xbee.devices.RemoteXBeeDevice
                                                            attribute), 213
        method), 798
                                                   discovery_status(digi.xbee.packets.common.TransmitStatusPacket
disable_bluetooth()
                                                            attribute), 287
        (digi.xbee.devices.RemoteZigBeeDevice
                                                                                                    in
                                                   DiscoveryOptions
                                                                                    (class
        method), 875
                                                            digi.xbee.models.options), 212
disable_bluetooth()
                                                    DiscoveryProcessFinished
                                                                                         (class
                                                                                                    in
        (digi.xbee.devices.WiFiDevice method), 773
                                                            digi.xbee.reader), 1002
disable bluetooth()
                                                   DiscoveryStatus (class in digi.xbee.models.status),
        (digi.xbee.devices.XBeeDevice
                                                            219
                                          method),
                                                   doc_enum() (in module digi.xbee.util.utils), 467
disable_bluetooth()
                                                   {\tt DONT\_ATTEMPT\_RD}~(digi.xbee.models.options. TransmitOptions
        (digi.xbee.devices.ZigBeeDevice
                                                            attribute), 210
                                          method),
                                                    dst_addr(digi.xbee.packets.digimesh.RouteInformationPacket
                                                            attribute), 328
disable_logger() (in module digi.xbee.util.utils),
                                                    E
DISABLE_RETRIES_AND_REPAIR
        (digi.xbee.models.options.TransmitOptions
                                                   EmberBootloaderMessageType
                                                                                          (class
                                                                                                    in
        attribute), 210
                                                            digi.xbee.models.status), 228
DISABLED (digi.xbee.io.IOMode attribute), 985
                                                    empty() (digi.xbee.reader.XBeeQueue method), 1026
disconnect()
                       (digi.xbee.devices.WiFiDevice
                                                   enable_apply_changes()
        method), 765
                                                            (digi.xbee.devices.AbstractXBeeDevice
disconnect() (digi.xbee.filesystem.LocalXBeeFileSystemManagermethod), 479
        method), 972
                                                    enable_apply_changes()
discover_device()
                                                            (digi.xbee.devices.CellularDevice
                                                                                              method),
        (digi.xbee.devices.DigiMeshNetwork method),
        932
                                                   enable_apply_changes()
discover device()
                                                            (digi.xbee.devices.DigiMeshDevice
                                                                                              method),
        (digi.xbee.devices.DigiPointNetwork method),
                                                            560
                                                    enable apply changes ()
discover_device()
                                                            (digi.xbee.devices.DigiPointDevice
                                                                                              method),
        (digi.xbee.devices.Raw802Network
                                          method),
                                                            593
        920
                                                    enable_apply_changes()
discover_device()
                                                            (digi.xbee.devices.IPDevice method), 663
        (digi.xbee.devices.XBeeNetwork
                                          method),
                                                    enable_apply_changes()
        893
                                                            (digi.xbee.devices.LPWANDevice
                                                                                              method),
discover_device()
                                                            714
        (digi.xbee.devices.ZigBeeNetwork
                                          method),
                                                   enable_apply_changes()
        908
                                                            (digi.xbee.devices.NBIoTDevice
                                                                                              method),
```

741	(digi.xbee.devices.RemoteDigiMeshDevice
<pre>enable_apply_changes()</pre>	method), 835
(digi.xbee.devices.Raw802Device method),	
527	(digi.xbee.devices.RemoteDigiPointDevice
<pre>enable_apply_changes()     (digi.xbee.devices.RemoteDigiMeshDevice</pre>	<pre>method), 854 enable_bluetooth()</pre>
method), 835	(digi.xbee.devices.RemoteRaw802Device
enable_apply_changes()	method), 816
(digi.xbee.devices.RemoteDigiPointDevice	enable_bluetooth()
method), 854	(digi.xbee.devices.RemoteXBeeDevice
enable_apply_changes()	method), 799
(digi.x bee. devices. Remote Raw 802 Device	<pre>enable_bluetooth()</pre>
method), 816	(digi.x bee. devices. Remote Zig Bee Device
<pre>enable_apply_changes()</pre>	method), 875
(digi.xbee.devices.RemoteXBeeDevice	enable_bluetooth()
method), 798	(digi.xbee.devices.WiFiDevice method), 774
enable_apply_changes()	enable_bluetooth()
(digi.xbee.devices.RemoteZigBeeDevice method), 875	(digi.xbee.devices.XBeeDevice method), 504
enable_apply_changes()	enable_bluetooth()
(digi.xbee.devices.WiFiDevice method), 774	(digi.xbee.devices.ZigBeeDevice method),
enable_apply_changes()	631
(digi.xbee.devices.XBeeDevice method), 504	<pre>enable_logger() (in module digi.xbee.util.utils), 467</pre>
enable_apply_changes()	ENABLE_MULTICAST (digi.xbee.models.options.TransmitOption
(digi.xbee.devices.ZigBeeDevice method),	attribute), 210
631	ENABLE_TRACE_ROUTE
ENABLE_APS_ENCRYPTION	(digi.x bee.models.options. Transmit Options
(digi.xbee.models.options.TransmitOptions	attribute), 210
attribute), 211	ENABLE_UNICAST_NACK
enable_bluetooth()	(digi.xbee.models.options.TransmitOptions
(digi.xbee.devices.AbstractXBeeDevice method), 488	attribute), 210 ENABLE_UNICAST_TRACE_ROUTE
enable_bluetooth()	(digi.xbee.models.options.TransmitOptions
(digi.xbee.devices.CellularDevice method),	attribute), 210
(anguino centre en central 2 evice memoa);	encryption_type (digi.xbee.models.accesspoint.AccessPoint
enable_bluetooth()	attribute), 130
(digi.xbee.devices.DigiMeshDevice method),	EndDiscoveryScan (class in digi.xbee.reader), 1014
560	<pre>enter_at_command_mode()</pre>
<pre>enable_bluetooth()</pre>	digi.xbee.recovery), 1028
(digi.xbee.devices.DigiPointDevice method),	error (digi.xbee.models.zdo.NeighborFinder attribute),
593	241
<pre>enable_bluetooth() (digi.xbee.devices.IPDevice</pre>	error (digi.xbee.models.zdo.NeighborTableReader attribute), 239
<pre>enable_bluetooth()</pre>	error (digi.xbee.models.zdo.NodeDescriptorReader at-
(digi.xbee.devices.LPWANDevice method),	tribute), 233
715	error (digi.xbee.models.zdo.RouteTableReader at-
enable_bluetooth()	tribute), 236
(digi.xbee.devices.NBIoTDevice method), 741	error (digi.xbee.packets.devicecloud.FrameErrorPacket attribute), 316
enable_bluetooth()	EXCLUSIVE (digi.xbee.models.options.FileOpenRequestOption
(digi.xbee.devices.Raw802Device method),	attribute), 215
527	exec_at_cmd() (digi.xbee.firmware.UpdateConfigurer
<pre>enable bluetooth()</pre>	static method), 976

<pre>execute_command()</pre>	method), 933
(digi.xbee.devices.AbstractXBeeDevice method), 475	export() (digi.xbee.devices.DigiPointNetwork method), 945
execute_command()	export () (digi.xbee.devices.Raw802Network method),
(digi.xbee.devices.CellularDevice method),	921
	export() (digi.xbee.devices.XBeeNetwork method),
<pre>execute_command()      (digi.xbee.devices.DigiMeshDevice method), 561</pre>	895 export() (digi.xbee.devices.ZigBeeNetwork method), 909
execute_command()	extend() (digi.xbee.reader.BluetoothDataReceived
(digi.xbee.devices.DigiPointDevice method),	method), 1006
593	extend() (digi.xbee.reader.DataReceived method),
execute_command() (digi.xbee.devices.IPDevice method), 663	998 extend() (digi.xbee.reader.DeviceDiscovered method),
<pre>execute_command()</pre>	1001
(digi.xbee.devices.LPWANDevice method), 715	extend() (digi.xbee.reader.DiscoveryProcessFinished method), 1002
<pre>execute_command()</pre>	extend() (digi.xbee.reader.EndDiscoveryScan
(digi.xbee.devices.NBIoTDevice method),	method), 1014
741	extend() (digi.xbee.reader.ExplicitDataReceived
execute_command()	method), 1003
(digi.xbee.devices.Raw802Device method),	extend() (digi.xbee.reader.FileSystemFrameReceived
528	method), 1015
execute_command()	extend() (digi.xbee.reader.InitDiscoveryScan
(digi.xbee.devices.RemoteDigiMeshDevice	method), 1014
method), 836	
execute_command()	extend() (digi.xbee.reader.IOSampleReceived method), 1000
(digi.xbee.devices.RemoteDigiPointDevice method), 854	extend() (digi.xbee.reader.IPDataReceived method), 1004
execute_command()	extend() (digi.xbee.reader.MicroPythonDataReceived
(digi.xbee.devices.RemoteRaw802Device method), 816	method), 1007 extend() (digi.xbee.reader.ModemStatusReceived
execute_command()	method), 999
(digi.xbee.devices.RemoteXBeeDevice	extend() (digi.xbee.reader.NetworkModified method),
method), 799	1001
execute_command()	extend() (digi.xbee.reader.NetworkUpdateProgress
(digi.xbee.devices.RemoteZigBeeDevice	method), 1016
method), 875	extend() (digi.xbee.reader.PacketReceived method),
execute_command() (digi.xbee.devices.WiFiDevice	996
method), 774	extend() (digi.xbee.reader.PacketReceivedFrom
execute_command() (digi.xbee.devices.XBeeDevice	method), 997
method), 504	extend() (digi.xbee.reader.RelayDataReceived
<pre>execute_command()</pre>	method), 1006
(digi.xbee.devices.ZigBeeDevice method), 631	extend() (digi.xbee.reader.RouteInformationReceived method), 1012
ExplicitAddressingPacket (class in digi.xbee.packets.common), 296	extend() (digi.xbee.reader.RouteReceived method), 1013
ExplicitDataReceived (class in digi.xbee.reader),	extend() (digi.xbee.reader.RouteRecordIndicatorReceived
1003	method), 1011
ExplicitRXIndicatorPacket (class in	extend() (digi.xbee.reader.SMSReceived method),
digi.xbee.packets.common), 301	1005
ExplicitXBeeMessage (class in	extend() (digi.xbee.reader.SocketDataReceived
digi.xbee.models.message), 204	method), 1009
export () (digi.xbee.devices.DigiMeshNetwork	<pre>extend() (digi.xbee.reader.SocketDataReceivedFrom</pre>

method), 1010 extend() (digi.xbee.reader.SocketStateReceived	flush_queues() (digi.xbee.devices.DigiPointDevice method), 594
method), 1008	flush_queues() (digi.xbee.devices.IPDevice
extend() (digi.xbee.reader.XBeeEvent method), 995	method), 664
EXTENDED_TIMEOUT (digi.xbee.models.options.RemoteAattribute), 212	AFCm&Qptionsues() (digi.xbee.devices.LPWANDevice method), 715
	flush_queues() (digi.xbee.devices.NBIoTDevice
F	method), 742
<pre>file_data(digi.xbee.packets.devicecloud.SendDataReq</pre>	นะ์ร่า <b>P</b> ลังหะคุนeues () (digi.xbee.devices.Raw802Device method), 528
file_hash(digi.xbee.models.filesystem.HashFileCmdReattribute), 159	sponseh_queues() (digi.xbee.devices.WiFiDevice method), 775
file_system_path (digi.xbee.profile.XBeeProfile at- tribute), 993	flush_queues() (digi.xbee.devices.XBeeDevice method), 497
FileIdCmd (class in digi.xbee.models.filesystem), 139	flush_queues() (digi.xbee.devices.ZigBeeDevice
FileIdNameCmd (class in digi.xbee.models.filesystem),	method), 632
141	force_disassociate()
FileOpenRequestOption (class in	(digi.xbee.devices.RemoteZigBeeDevice
digi.xbee.models.options), 215	method), 872
FileProcess (class in digi.xbee.filesystem), 960	force_disassociate()
FileSystemElement (class in digi.xbee.filesystem), 958	(digi.xbee.devices.ZigBeeDevice method), 616
FileSystemException, 959	format (digi.xbee.profile.XBeeProfileSetting attribute),
FileSystemFrameReceived (class in	989
digi.xbee.reader), 1015	format() (digi.xbee.filesystem.FileSystemManager
FileSystemManager (class in digi.xbee.filesystem), 960	<pre>method), 965 format_filesystem()</pre>
FileSystemNotSupportedException, 959	(digi.xbee.file system. Local XBee File System Manager
finished(digi.xbee.models.status.UpdateProgressStatus	
attribute), 233	frame_id (digi.xbee.packets.base.GenericXBeePacket
firmware_description_file	attribute), 249
(digi.xbee.profile.XBeeProfile attribute), 993	frame_id(digi.xbee.packets.base.UnknownXBeePacket attribute), 252
firmware_version (digi.xbee.profile.XBeeProfile attribute), 992	<pre>frame_id (digi.xbee.packets.base.XBeeAPIPacket at- tribute), 247</pre>
FirmwareBaudrate (class in digi.xbee.profile), 985	frame_id (digi.xbee.packets.cellular.RXSMSPacket at-
FirmwareParity (class in digi.xbee.profile), 986	tribute), 255
FirmwareStopbits (class in digi.xbee.profile), 986	<pre>frame_id (digi.xbee.packets.cellular.TXSMSPacket at-</pre>
FirmwareUpdateException, 958	tribute), 257
flags (digi.xbee.packets.devicecloud.DeviceRequestPackets), 307	efframe_id (digi.xbee.packets.common.ATCommPacket attribute), 261
flash_firmware_option	<pre>frame_id(digi.xbee.packets.common.ATCommQueuePacket</pre>
(digi.xbee.profile.XBeeProfile attribute),	attribute), 264
991	frame_id(digi.xbee.packets.common.ATCommResponsePacket
FlashFirmwareOption (class in digi.xbee.profile),	attribute), 268
987	frame_id(digi.xbee.packets.common.ExplicitAddressingPacket
FLOOD (digi.xbee.models.mode.NeighborDiscoveryMode attribute), 199	<pre>attribute), 298 frame_id(digi.xbee.packets.common.ExplicitRXIndicatorPacket</pre>
FlowControl (class in digi.xbee.serial), 1030	attribute), 302
flush() (digi.xbee.reader.XBeeQueue method), 1028	<pre>frame_id(digi.xbee.packets.common.IODataSampleRxIndicatorPack</pre>
flush_queues() (digi.xbee.devices.CellularDevice	attribute), 294
method), 690	frame_id(digi.xbee.packets.common.ModemStatusPacket
flush_queues() (digi.xbee.devices.DigiMeshDevice method), 561	attribute), 290 frame_id (digi.xbee.packets.common.ReceivePacket
тетоа), эот	(anomination in the control which

attribute), 271	attribute), 414
frame_id(digi.xbee.packets.common.RemoteATCommandPacket	
attribute), 275	attribute), 401
frame_id (digi.xbee.packets.common.RemoteATCommandResperattribute), 279	<u>sseBalahen.xbee.packets.socket.SocketCloseResponsePacket</u> attribute), 404
<pre>frame_id (digi.xbee.packets.common.TransmitPacket frame</pre>	
attribute), 284	attribute), 395
<pre>frame_id(digi.xbee.packets.common.TransmitStatusPacketrame</pre>	
frame_id (digi.xbee.packets.devicecloud.DeviceRequestPacketme	
attribute), 307	attribute), 381
<pre>frame_id (digi.xbee.packets.devicecloud.DeviceResponsePackete</pre>	_id (digi.xbee.packets.socket.SocketCreateResponsePacket attribute), 384
frame_id(digi.xbee.packets.devicecloud.DeviceResponseStatus)	
attribute), 313	attribute), 417
frame_id(digi.xbee.packets.devicecloud.FrameErrorPack@trame	
attribute), 316	attribute), 420
frame_id(digi.xbee.packets.devicecloud.SendDataRequestPacke	
attribute), 319	attribute), 388
frame_id (digi.xbee.packets.devicecloud.SendDataResportseEarch attribute), 322	<u>ettad (digi.xbee.packets.socket.SocketOptionResponsePacket</u> attribute), 391
frame_id(digi.xbee.packets.digimesh.RouteInformationPacketme	
attribute), 326	_1a (aigi.xbee.packeis.sockei.sockeikeceiverromrackei attribute), 426
<pre>frame_id(digi.xbee.packets.filesystem.FSRequestPacket frame</pre>	
attribute), 330	attribute), 423
<pre>frame_id(digi.xbee.packets.filesystem.FSResponsePacketframe</pre>	_id (digi.xbee.packets.socket.SocketSendPacket attribute), 407
frame_id (digi.xbee.packets.filesystem.RemoteFSRequestPackete	
attribute), 337	attribute), 411
frame_id(digi.xbee.packets.filesystem.RemoteFSRespons&Rockets	
attribute), 339	attribute), 429
	_id(digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacket
tribute), 344	attribute), 433
· · ·	_id (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacke
tribute), 347	attribute), 440
<pre>frame_id (digi.xbee.packets.raw.RX16IOPacket frame</pre>	_id(digi.xbee.packets.wifi.RemoteATCommandWifiPacket attribute), 437
frame_id (digi.xbee.packets.raw.RX16Packet at- frame	
tribute), 364	attribute), 456
<pre>frame_id (digi.xbee.packets.raw.RX64IOPacket frame</pre>	_id (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket attribute), 458
	_id(digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
tribute), 361	attribute), 448
frame_id (digi.xbee.packets.raw.TX16Packet at- frame	_id(digi.xbee.packets.zigbee.RegisterJoiningDevicePacket
tribute), 354	attribute), 445
<pre>frame_id (digi.xbee.packets.raw.TX64Packet at- frame</pre>	_id (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket attribute), 452
	Error (class in digi.xbee.models.status), 226
	ErrorPacket (class in
frame_id (digi.xbee.packets.relay.UserDataRelayOutputPacket	digi.xbee.packets.devicecloud), 315
	band (digi.xbee.models.zdo.NodeDescriptor at-
frame_id(digi.xbee.packets.relay.UserDataRelayPacket	tribute), 234
	bytes() (digi.xbee.models.address.XBee16BitAddress
frame id(digi.xbee.packets.socket.SocketBindListenPacket	

<pre>from_bytes() (digi.xbee.models.address.XBee64BitAdd</pre>	hess_id(digi.xbee.models.filesystem.WriteFileCmdResponse attribute), 156
<pre>from_data() (digi.xbee.filesystem.FileSystemElement</pre>	FSCmd (class in digi.xbee.models.filesystem), 136 FSCmdType (class in digi.xbee.models.filesystem), 136
<pre>from_hex_string()</pre>	FSCommandStatus (class in digi.xbee.models.status),
(digi.x bee.models. address. XBee 16 Bit Address	231
class method), 201	FSRequestPacket (class in
<pre>from_hex_string()</pre>	digi.xbee.packets.filesystem), 329
(digi.x bee.models. address. XBee 64Bit Address	FSResponsePacket (class in
class method), 202	digi.xbee.packets.filesystem), 332
<pre>from_string() (digi.xbee.models.address.XBeeIMEIAc</pre>	
class method), 203	$\verb full_path  (\textit{digi.xbee.models.filesystem.GetPathIdCmdResponsed)    \textit{full_path}  (\textit{digi.xbee.models.filesystem.GetPathIdCmdResponsed)      \textit{full_path}  (digi.xbee.models.filesystem.GetPathIdCmdResponsed)                                      $
$\verb fs_entries  (digi.x bee.models. filesystem. Open Dir CmdR)  \\$	-
attribute), 166	<pre>fw_path (digi.xbee.firmware.FwUpdateTask attribute),</pre>
$\verb fs_entries  (digi.x bee.models. filesystem. Read Dir CmdRead D$	
attribute), 173	FwUpdateTask (class in digi.xbee.firmware), 977
$\verb fs_id  (digi.xbee.models.filesystem.CloseDirCmdRequest $	
attribute), 168	G
$\verb fs_id  (digi.x bee.models.filesystem.CloseFileCmdRequest $	<pre>generate_network_xml() (in module</pre>
attribute), 147	digi.xbee.util.exportutils), 463
$\verb fs_id  (digi.x bee.models.filesystem.CreateDirCmdRequest) $	GenericXBeePacket (class in
attribute), 161	digi.xbee.packets.base), 248
<pre>fs_id (digi.xbee.models.filesystem.DeleteCmdRequest</pre>	get () (digi.xbee.reader.XBeeQueue method), 1025
attribute), 182	<pre>get_16bit_addr() (digi.xbee.devices.AbstractXBeeDevice</pre>
fs_id (digi.xbee.models.filesystem.FileIdCmd at-	method), 478
tribute), 140	<pre>get_16bit_addr() (digi.xbee.devices.CellularDevice</pre>
<pre>fs_id (digi.xbee.models.filesystem.FileIdNameCmd at-</pre>	method), 690
tribute), 142	<pre>get_16bit_addr() (digi.xbee.devices.DigiMeshDevice</pre>
$\verb fs_id  (digi.xbee.models.filesystem.GetPathIdCmdRequest) $	method), 561
attribute), 175	<pre>get_16bit_addr() (digi.xbee.devices.DigiPointDevice</pre>
$\verb fs_id  (digi.xbee.models.filesystem.GetPathIdCmdResponser)  \\$	se method), 594
attribute), 177	<pre>get_16bit_addr() (digi.xbee.devices.IPDevice</pre>
$\verb fs_id  (digi.xbee.models.filesystem.HashFileCmdRequest $	method), 657
attribute), 158	<pre>get_16bit_addr() (digi.xbee.devices.LPWANDevice</pre>
fs_id(digi.xbee.models.filesystem.OpenDirCmdRequest	method), 715
attribute), 165	<pre>get_16bit_addr() (digi.xbee.devices.NBIoTDevice</pre>
fs_id(digi.xbee.models.filesystem.OpenDirCmdResponse	method), 742
attribute), 167	<pre>get_16bit_addr() (digi.xbee.devices.Raw802Device</pre>
$\verb fs_id  (digi.xbee.models.filesystem.OpenFileCmdRequest ) \\$	method), 528
attribute), 144	$\verb"get_16bit_addr" () \textit{ (digi.xbee.devices.RemoteDigiMeshDevices)} \\$
$\verb fs_id  (digi.xbee.models.filesystem.OpenFileCmdResponse) $	memou), occ
attribute), 146	<pre>get_16bit_addr() (digi.xbee.devices.RemoteDigiPointDevices</pre>
fs_id(digi.xbee.models.filesystem.ReadDirCmdRequest	method), 855
attribute), 171	<pre>get_16bit_addr() (digi.xbee.devices.RemoteRaw802Device</pre>
fs_id(digi.xbee.models.filesystem.ReadDirCmdResponse	memou), or
attribute), 173	<pre>get_16bit_addr() (digi.xbee.devices.RemoteXBeeDevice</pre>
fs_id(digi.xbee.models.filesystem.ReadFileCmdRequest	method), 799
attribute), 151	<pre>get_16bit_addr() (digi.xbee.devices.RemoteZigBeeDevice</pre>
fs_id(digi.xbee.models.filesystem.ReadFileCmdResponse	,,
attribute), 153	get_16bit_addr() (digi.xbee.devices.WiFiDevice
fs_id (digi.xbee.models.filesystem.RenameCmdRequest	method), 775
attribute), 179	get_16bit_addr() (digi.xbee.devices.XBeeDevice
fs_id(digi.xbee.models.filesystem.WriteFileCmdRequest attribute), 155	method), 505

```
qet_16bit_addr() (digi.xbee.devices.ZigBeeDevice qet_adc_value() (digi.xbee.devices.RemoteDigiMeshDevice
        method), 632
                                                            method), 837
get_64bit_addr() (digi.xbee.devices.AbstractXBeeDevicet_adc_value() (digi.xbee.devices.RemoteDigiPointDevice
                                                            method), 855
        method), 478
get_64bit_addr() (digi.xbee.devices.CellularDevice get_adc_value() (digi.xbee.devices.RemoteRaw802Device
        method), 684
                                                            method), 817
get 64bit addr() (digi.xbee.devices.DigiMeshDeviceget add value() (digi.xbee.devices.RemoteXBeeDevice
        method), 562
                                                            method), 800
get_64bit_addr() (digi.xbee.devices.DigiPointDevice get_adc_value() (digi.xbee.devices.RemoteZigBeeDevice
        method), 594
                                                            method), 876
get_64bit_addr()
                         (digi.xbee.devices.IPDevice get_adc_value()
                                                                           (digi.xbee.devices.WiFiDevice
        method), 664
                                                            method), 775
get_64bit_addr() (digi.xbee.devices.LPWANDevice get_adc_value()
                                                                          (digi.xbee.devices.XBeeDevice
                                                            method), 505
        method), 716
get_64bit_addr() (digi.xbee.devices.NBIoTDevice get_adc_value()
                                                                         (digi.xbee.devices.ZigBeeDevice
        method), 742
                                                            method), 632
qet_64bit_addr() (digi.xbee.devices.Raw802Device qet_ai_status() (digi.xbee.devices.Raw802Device
        method), 529
                                                            method), 520
get_64bit_addr() (digi.xbee.devices.RemoteDigiMeshthericai_status() (digi.xbee.devices.RemoteRaw802Device
        method), 836
                                                            method), 814
get_64bit_addr() (digi.xbee.devices.RemoteDigiPointDeviceai_status() (digi.xbee.devices.RemoteZigBeeDevice
        method), 855
                                                            method), 872
get_64bit_addr() (digi.xbee.devices.RemoteRaw802Dgreite_ai_status()
                                                                        (digi.xbee.devices.ZigBeeDevice
        method), 817
                                                            method), 616
get_64bit_addr() (digi.xbee.devices.RemoteXBeeDeviget_analog_value()
                                                                                 (digi.xbee.io.IOSample
        method), 800
                                                            method), 984
get_64bit_addr() (digi.xbee.devices.RemoteZigBeeDegiere_api_output_mode()
                                                            (digi.xbee.devices.AbstractXBeeDevice
        method), 876
get_64bit_addr()
                       (digi.xbee.devices.WiFiDevice
                                                            method), 486
        method), 775
                                                   get_api_output_mode()
get_64bit_addr()
                      (digi.xbee.devices.XBeeDevice
                                                            (digi.xbee.devices.CellularDevice
                                                                                             method),
        method), 505
                                                            691
get_64bit_addr() (digi.xbee.devices.ZigBeeDevice
                                                   get_api_output_mode()
        method), 632
                                                            (digi.xbee.devices.DigiMeshDevice
                                                                                             method),
get_access_point()
        (digi.xbee.devices.WiFiDevice method), 763
                                                   get_api_output_mode()
get_access_point_timeout()
                                                            (digi.xbee.devices.DigiPointDevice
                                                                                             method),
        (digi.xbee.devices.WiFiDevice method), 766
get_adc_value() (digi.xbee.devices.AbstractXBeeDevicet_api_output_mode()
                                                            (digi.xbee.devices.IPDevice method), 664
        method), 483
get_adc_value() (digi.xbee.devices.CellularDevice
                                                   get_api_output_mode()
        method), 690
                                                            (digi.xbee.devices.LPWANDevice
                                                                                             method),
get_adc_value() (digi.xbee.devices.DigiMeshDevice
        method), 562
                                                   get_api_output_mode()
get_adc_value() (digi.xbee.devices.DigiPointDevice
                                                            (digi.xbee.devices.NBIoTDevice
                                                                                             method),
        method), 594
                                                            742
get_adc_value()
                         (digi.xbee.devices.IPDevice
                                                   get_api_output_mode()
        method), 664
                                                            (digi.xbee.devices.Raw802Device
                                                                                             method),
get_adc_value() (digi.xbee.devices.LPWANDevice
                                                            529
        method), 716
                                                   get_api_output_mode()
                     (digi.xbee.devices.NBIoTDevice
                                                            (digi.xbee.devices.RemoteDigiMeshDevice
get_adc_value()
        method), 742
                                                            method), 837
get_adc_value() (digi.xbee.devices.Raw802Device
                                                   get_api_output_mode()
        method), 529
                                                            (digi.xbee.devices.RemoteDigiPointDevice
```

	pd), 856		(digi.xbee.devices.RemoteZigBeeDevice	2
	xbee.devices.RemoteRaw802De	vice	<pre>method), 877 get_api_output_mode_value()</pre>	
	od), 818		(digi.xbee.devices.WiFiDevice method),	776
	tput_mode() xbee.devices.RemoteXBeeDevic od), 800	e	<pre>get_api_output_mode_value()     (digi.xbee.devices.XBeeDevice</pre>	nethod),
get_api_out (digi		ice	<pre>get_api_output_mode_value()     (digi.xbee.devices.ZigBeeDevice</pre>	nethod),
get_api_out (digi	tput_mode () xbee.devices.WiFiDevice method	d), 775	<pre>get_bluetooth_data_received_callb     (digi.xbee.reader.PacketListener method</pre>	
get_api_out (digi 506	tput_mode() xbee.devices.XBeeDevice	method),	<pre>get_bluetooth_mac_addr()     (digi.xbee.devices.AbstractXBeeDevice     method), 488</pre>	
get_api_out (digi	tput_mode() xbee.devices.ZigBeeDevice	method),	get_bluetooth_mac_addr() (digi.xbee.devices.CellularDevice n	nethod),
(digi	tput_mode_value() xbee.devices.AbstractXBeeDevic od), 487	ce	<pre>get_bluetooth_mac_addr()     (digi.xbee.devices.DigiMeshDevice</pre>	nethod),
	tput_mode_value() xbee.devices.CellularDevice	method),	get_bluetooth_mac_addr() (digi.xbee.devices.DigiPointDevice n 596	nethod),
	tput_mode_value() xbee.devices.DigiMeshDevice	method),	<pre>get_bluetooth_mac_addr()           (digi.xbee.devices.IPDevice method), 6 get_bluetooth_mac_addr()</pre>	65
	tput_mode_value() xbee.devices.DigiPointDevice	method),	(digi.xbee.devices.LPWANDevice r. 717	nethod),
get_api_out	tput_mode_value() xbee.devices.IPDevice method),	665	get_bluetooth_mac_addr() (digi.xbee.devices.NBIoTDevice 743	nethod),
	tput_mode_value() xbee.devices.LPWANDevice	method),	get_bluetooth_mac_addr() (digi.xbee.devices.Raw802Device n 530	nethod),
	tput_mode_value() xbee.devices.NBIoTDevice	method),	<pre>get_bluetooth_mac_addr()      (digi.xbee.devices.RemoteDigiMeshDev</pre>	rice
	tput_mode_value() xbee.devices.Raw802Device	method),	<pre>get_bluetooth_mac_addr()      (digi.xbee.devices.RemoteDigiPointDev</pre>	rice
(digi	tput_mode_value() xbee.devices.RemoteDigiMeshD od),837	<b>)</b> evice	<pre>get_bluetooth_mac_addr()      (digi.xbee.devices.RemoteRaw802Device method), 819</pre>	ce
get_api_out (digi	tput_mode_value() xbee.devices.RemoteDigiPointD od), 856	)evice	<pre>get_bluetooth_mac_addr()      (digi.xbee.devices.RemoteXBeeDevice      method), 801</pre>	
get_api_out (digi	tput_mode_value()  xbee.devices.RemoteRaw802De  od), 818	vice	get_bluetooth_mac_addr() (digi.xbee.devices.RemoteZigBeeDevice method), 878	2
get_api_out (digi	tput_mode_value()  xbee.devices.RemoteXBeeDeviced), 801	e	<pre>get_bluetooth_mac_addr()             (digi.xbee.devices.WiFiDevice method), get_bluetooth_mac_addr()</pre>	776
	tput_mode_value()			nethod),

```
507
                                                                                               method), 287
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.DeviceRequestPacket
get_bluetooth_mac_addr()
             (digi.xbee.devices.ZigBeeDevice
                                                                  method),
                                                                                               method), 307
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.DeviceResponsePacket
get_by_id() (digi.xbee.reader.XBeeQueue method),
                                                                                               method), 310
                                                                                  get checksum() (digi.xbee.packets.devicecloud.DeviceResponseStatus.
             1027
get_by_ip() (digi.xbee.reader.XBeeQueue method),
                                                                                               method), 313
             1026
                                                                                  get_checksum() (digi.xbee.packets.devicecloud.FrameErrorPacket
get_by_remote()
                                      (digi.xbee.reader.XBeeQueue
                                                                                               method), 316
             method), 1026
                                                                                  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.SendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendDataRequestSendD
get_cellular_ai_status()
                                                                                               method), 320
             (digi.xbee.devices.CellularDevice
                                                                                 get_checksum() (digi.xbee.packets.devicecloud.SendDataResponsePac
                                                                  method),
                                                                                               method), 322
                                                                                 get_checksum() (digi.xbee.packets.digimesh.RouteInformationPacket
get_cellular_ai_status()
             (digi.xbee.devices.LPWANDevice
                                                                  method),
                                                                                               method), 326
             717
                                                                                  get_checksum() (digi.xbee.packets.filesystem.FSRequestPacket
                                                                                               method), 330
get_cellular_ai_status()
             (digi.xbee.devices.NBIoTDevice
                                                                                 get_checksum() (digi.xbee.packets.filesystem.FSResponsePacket
                                                                  method),
                                                                                               method), 333
get checksum() (digi.xbee.packets.base.GenericXBeePacket checksum() (digi.xbee.packets.filesystem.RemoteFSRequestPacket
             method), 249
                                                                                               method), 337
get_checksum() (digi.xbee.packets.base.UnknownXBeeBacketchecksum() (digi.xbee.packets.filesystem.RemoteFSResponsePacket
             method), 252
                                                                                               method), 339
get checksum() (digi.xbee.packets.base.XBeeAPIPacketget checksum() (digi.xbee.packets.network.RXIPv4Packet
             method), 247
                                                                                               method), 344
get_checksum() (digi.xbee.packets.base.XBeePacket get_checksum() (digi.xbee.packets.network.TXIPv4Packet
                                                                                               method), 347
             method), 244
get_checksum() (digi.xbee.packets.cellular.RXSMSPackget_checksum() (digi.xbee.packets.raw.RX16IOPacket
             method), 255
                                                                                               method), 370
get_checksum() (digi.xbee.packets.cellular.TXSMSPackget_checksum() (digi.xbee.packets.raw.RX16Packet
             method), 258
                                                                                               method), 364
get_checksum() (digi.xbee.packets.common.ATCommPacket_checksum() (digi.xbee.packets.raw.RX64IOPacket
             method), 261
                                                                                               method), 368
get_checksum() (digi.xbee.packets.common.ATCommQyettePableecksum() (digi.xbee.packets.raw.RX64Packet
                                                                                               method), 361
             method), 264
get_checksum() (digi.xbee.packets.common.ATCommRespon.satPacketsum() (digi.xbee.packets.raw.TX16Packet
             method), 268
                                                                                               method), 354
get_checksum() (digi.xbee.packets.common.ExplicitAddyexsingPacketsum() (digi.xbee.packets.raw.TX64Packet
                                                                                               method), 351
             method), 298
get_checksum() (digi.xbee.packets.common.ExplicitRXhpdicatorPacketm() (digi.xbee.packets.raw.TXStatusPacket
             method), 302
                                                                                               method), 357
get_checksum() (digi.xbee.packets.common.IODataSampleRxIndicatovRackddigi.xbee.packets.relay.UserDataRelayOutputPacket
             method), 295
                                                                                               method), 378
get_checksum() (digi.xbee.packets.common.ModemStatysPacketecksum() (digi.xbee.packets.relay.UserDataRelayPacket
             method), 290
                                                                                               method), 375
get_checksum() (digi.xbee.packets.common.ReceivePacket_checksum() (digi.xbee.packets.socket.SocketBindListenPacket
             method), 271
                                                                                               method), 414
get_checksum() (digi.xbee.packets.common.RemoteATGentumaindRakketm() (digi.xbee.packets.socket.SocketClosePacket
             method), 275
                                                                                               method), 401
get_checksum() (digi.xbee.packets.common.RemoteATGornmandReisponsetfloddigi.xbee.packets.socket.SocketCloseResponsePacket
             method), 279
                                                                                               method), 404
get_checksum() (digi.xbee.packets.common.TransmitPacket_checksum() (digi.xbee.packets.socket.SocketConnectPacket
             method), 284
                                                                                               method), 395
get checksum() (digi.xbee.packets.common.TransmitStateuxPacketcksum() (digi.xbee.packets.socket.SocketConnectResponsePack
```

```
method), 398
                                                             945
get checksum() (digi.xbee.packets.socket.SocketCreateBacketconnections()
        method), 381
                                                             (digi.xbee.devices.Raw802Network
                                                                                               method),
get_checksum() (digi.xbee.packets.socket.SocketCreateResponsePacket
        method), 384
                                                    get connections()
get checksum() (digi.xbee.packets.socket.SocketListenResponsePadilgitxbee.devices.XBeeNetwork
                                                                                               method),
                                                             903
        method), 417
get_checksum() (digi.xbee.packets.socket.SocketNewIPv4ctlientPackettions()
        method), 420
                                                             (digi.xbee.devices.ZigBeeNetwork
                                                                                               method),
get_checksum() (digi.xbee.packets.socket.SocketOptionRequestPacket
        method), 388
                                                    get_current_directory()
qet_checksum() (digi.xbee.packets.socket.SocketOptionResponsePdigiexbee.filesystem.LocalXBeeFileSystemManager
                                                             method), 972
        method), 391
get_checksum() (digi.xbee.packets.socket.SocketReceiveFromPacketent_frame_id()
                                                             (digi.xbee.devices.AbstractXBeeDevice
        method), 426
get_checksum() (digi.xbee.packets.socket.SocketReceivePacket
                                                            method), 479
        method), 423
                                                    get_current_frame_id()
get_checksum() (digi.xbee.packets.socket.SocketSendPacket
                                                             (digi.xbee.devices.CellularDevice
                                                                                               method),
        method), 407
get checksum() (digi.xbee.packets.socket.SocketSendToPacketurrent frame id()
        method), 411
                                                             (digi.xbee.devices.DigiMeshDevice
                                                                                               method),
get_checksum() (digi.xbee.packets.socket.SocketStatePacket
                                                    get_current_frame_id()
        method), 429
get_checksum() (digi.xbee.packets.wifi.IODataSampleRxIndicatonWifiRbekadevices.DigiPointDevice
                                                                                               method).
        method), 434
get_checksum() (digi.xbee.packets.wifi.RemoteATCommontdResponseWifiPackete_id()
        method), 440
                                                             (digi.xbee.devices.IPDevice method), 666
get_checksum() (digi.xbee.packets.wifi.RemoteATCommycetd_Wifi.Packett_frame_id()
        method), 437
                                                             (digi.xbee.devices.LPWANDevice
                                                                                               method),
get_checksum() (digi.xbee.packets.zigbee.CreateSourceRoutePacket8
        method), 456
                                                    get_current_frame_id()
get_checksum() (digi.xbee.packets.zigbee.OTAFirmwareUpdateS(digsRbbekedevices.NBIoTDevice
                                                                                               method),
        method), 458
                                                             744
get_checksum() (digi.xbee.packets.zigbee.RegisterDeviceStatusPackett_frame_id()
        method), 448
                                                             (digi.xbee.devices.Raw802Device
                                                                                               method),
get_checksum() (digi.xbee.packets.zigbee.RegisterJoiningDeviceEdcket
        method), 445
                                                    get current frame id()
get_checksum() (digi.xbee.packets.zigbee.RouteRecordIndicatorRdigexbee.devices.RemoteDigiMeshDevice
        method), 452
                                                             method), 838
get_comm_iface() (digi.xbee.devices.RemoteDigiMeshPericeurrent_frame_id()
                                                             (digi.xbee.devices.RemoteDigiPointDevice
        method), 838
get_comm_iface() (digi.xbee.devices.RemoteDigiPointDevice
                                                             method), 857
        method), 857
                                                    get_current_frame_id()
get_comm_iface() (digi.xbee.devices.RemoteRaw802Device
                                                             (digi.xbee.devices. Remote Raw 802 Device\\
        method), 819
                                                             method), 819
qet_comm_iface() (digi.xbee.devices.RemoteXBeeDeviqet_current_frame_id()
        method), 797
                                                             (digi.xbee.devices.RemoteXBeeDevice
get_comm_iface() (digi.xbee.devices.RemoteZigBeeDevice
                                                             method), 802
        method), 878
                                                    get_current_frame_id()
                                                             (digi.xbee.devices.RemoteZigBeeDevice
get_connections()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                             method), 878
        933
                                                    get current frame id()
get_connections()
                                                             (digi.xbee.devices.WiFiDevice method), 777
        (digi.xbee.devices.DigiPointNetwork method), get current frame id()
```

	(digi.xbee.devices.XBeeDevice 507	method),	<pre>get_dest_address()     (digi.xbee.devices.LPWANDevice</pre>	method),
get_c	current_frame_id()     (digi.xbee.devices.ZigBeeDevice 634	method),	<del>-</del>	m ath a d
get_c	data_queue() (digi.xbee.reader.Pack method), 1018	etListener	(digi.xbee.devices.NBIoTDevice 744 get_dest_address()	method),
get_c	data_received_callbacks() (digi.xbee.reader.PacketListener 1023	method),		method),
get_d	deep_discovery_options() (digi.xbee.devices.DigiMeshNetwork 934	method),	<pre>(digi.xbee.devices.RemoteDigiMeshDe</pre>	vice
get_c	deep_discovery_options() (digi.xbee.devices.DigiPointNetwork 946	method),	(digi.xbee.devices.RemoteDigiPointDemethod), 857 get_dest_address()	vice
get_d	deep_discovery_options() (digi.xbee.devices.Raw802Network 922	method),	(digi.xbee.devices.RemoteRaw802Devi method), 819 get_dest_address()	ce
get_c	deep_discovery_options() (digi.xbee.devices.XBeeNetwork 899	method),	(digi.xbee.devices.RemoteXBeeDevice method), 802 get_dest_address()	
get_d	deep_discovery_options() (digi.xbee.devices.ZigBeeNetwork 910	method),	(digi.xbee.devices.RemoteZigBeeDevicemethod), 878 get_dest_address()	e
get_c	deep_discovery_timeouts() (digi.xbee.devices.DigiMeshNetwork 934	method),	<pre>(digi.xbee.devices.WiFiDevice method) get_dest_address()</pre>	, 777 method),
get_c	deep_discovery_timeouts() (digi.xbee.devices.DigiPointNetwork 946	method),	507 get_dest_address()	method),
get_d	deep_discovery_timeouts() (digi.xbee.devices.Raw802Network 922	method),	634 get_dest_ip_addr()	method),
get_d	deep_discovery_timeouts() (digi.xbee.devices.XBeeNetwork 900	method),	<pre>692 get_dest_ip_addr() (digi.xbee.devices.if method), 654</pre>	IPDevice
get_d	deep_discovery_timeouts() (digi.xbee.devices.ZigBeeNetwork 910	method),	get_dest_ip_addr()	method),
get_c	<pre>dest_address()    (digi.xbee.devices.AbstractXBeeDevic    method), 479</pre>	ce	<pre>get_dest_ip_addr()      (digi.xbee.devices.NBIoTDevice</pre>	method),
get_d	dest_address() (digi.xbee.devices.CellularDevice 692	method),	<pre>get_dest_ip_addr()             (digi.xbee.devices.WiFiDevice method) get_device_by_16()</pre>	, 777
get_c	dest_address() (digi.xbee.devices.DigiMeshDevice 564	method),	(digi.xbee.devices.DigiMeshNetwork 934 get_device_by_16()	method),
get_c	dest_address()     (digi.xbee.devices.DigiPointDevice	method),	(digi.xbee.devices.DigiPointNetwork 946	method),
get_c	596 dest_address() (digi.xbee.devices method), 657	.IPDevice	<pre>get_device_by_16()     (digi.xbee.devices.Raw802Network 922</pre>	method),

```
get_device_by_16()
                                                   get_dio_value() (digi.xbee.devices.DigiPointDevice
        (digi.xbee.devices.XBeeNetwork
                                         method).
                                                            method), 596
                                                   get_dio_value()
        902
                                                                             (digi.xbee.devices.IPDevice
get_device_by_16()
                                                            method), 666
        (digi.xbee.devices.ZigBeeNetwork
                                         method),
                                                   get_dio_value() (digi.xbee.devices.LPWANDevice
                                                            method), 718
                                                                         (digi.xbee.devices.NBIoTDevice
get_device_by_64()
                                                   get_dio_value()
        (digi.xbee.devices.DigiMeshNetwork method),
                                                            method), 744
        935
                                                   get_dio_value() (digi.xbee.devices.Raw802Device
get_device_by_64()
                                                            method), 531
                                                   \verb"get_dio_value" () \textit{ (digi.xbee.devices.RemoteDigiMeshDevice)} \\
        (digi.xbee.devices.DigiPointNetwork method),
                                                            method), 839
get_device_by_64()
                                                   get_dio_value() (digi.xbee.devices.RemoteDigiPointDevice
        (digi.xbee.devices.Raw802Network
                                         method),
                                                            method), 858
        923
                                                   get_dio_value() (digi.xbee.devices.RemoteRaw802Device
get_device_by_64()
                                                            method), 820
        (digi.xbee.devices.XBeeNetwork
                                                   get_dio_value() (digi.xbee.devices.RemoteXBeeDevice
                                         method),
        902
                                                            method), 802
                                                   \verb"get_dio_value" () \textit{ (digi.xbee.devices.RemoteZigBeeDevice}
get_device_by_64()
        (digi.xbee.devices.ZigBeeNetwork
                                         method),
                                                            method), 879
        911
                                                   get_dio_value()
                                                                           (digi.xbee.devices.WiFiDevice
get_device_by_node_id()
                                                            method), 777
        (digi.xbee.devices.DigiMeshNetwork method),
                                                                          (digi.xbee.devices.XBeeDevice
                                                   get_dio_value()
        935
                                                            method), 507
get_device_by_node_id()
                                                   get_dio_value()
                                                                         (digi.xbee.devices.ZigBeeDevice
        (digi.xbee.devices.DigiPointNetwork method),
                                                            method), 635
                                                   get_discovery_callbacks()
                                                            (digi.xbee.devices.DigiMeshNetwork method),
get_device_by_node_id()
        (digi.xbee.devices.Raw802Network
                                         method),
        923
                                                   get_discovery_callbacks()
get_device_by_node_id()
                                                            (digi.xbee.devices.DigiPointNetwork method),
        (digi.xbee.devices.XBeeNetwork
                                         method),
                                                            947
                                                   get_discovery_callbacks()
        902
                                                            (digi.xbee.devices.Raw802Network
get_device_by_node_id()
                                                                                             method),
        (digi.xbee.devices.ZigBeeNetwork
                                         method),
                                                   get_discovery_callbacks()
get_devices() (digi.xbee.devices.DigiMeshNetwork
                                                            (digi.xbee.devices.XBeeNetwork
                                                                                             method),
        method), 935
                                                            903
get_devices() (digi.xbee.devices.DigiPointNetwork
                                                   get_discovery_callbacks()
                                                            (digi.xbee.devices.ZigBeeNetwork
        method), 947
                                                                                             method),
                  (digi.xbee.devices.Raw802Network
get_devices()
        method), 923
                                                   get_discovery_options()
get_devices()
                     (digi.xbee.devices.XBeeNetwork
                                                            (digi.xbee.devices.DigiMeshNetwork method),
        method), 894
                                                            935
get_devices()
                    (digi.xbee.devices.ZigBeeNetwork
                                                   get_discovery_options()
        method), 911
                                                            (digi.xbee.devices.DigiPointNetwork method),
get_digital_value()
                             (digi.xbee.io.IOSample
                                                            947
        method), 984
                                                   get_discovery_options()
get_dio_value() (digi.xbee.devices.AbstractXBeeDevice
                                                            (digi.xbee.devices.Raw802Network
                                                                                             method),
        method), 485
get_dio_value() (digi.xbee.devices.CellularDevice
                                                   get_discovery_options()
                                                            (digi.xbee.devices.XBeeNetwork
        method), 693
                                                                                             method),
get_dio_value() (digi.xbee.devices.DigiMeshDevice
        method), 564
                                                   get_discovery_options()
```

		531	
	g		
	4 D		
	g		
	athod)		
	9	<del>_</del>	
	ethod).		
	9		
	ethod),		
900			
_discovery_timeout()	_	(digi.xbee.devices.RemoteZigBeeDevice	
(digi.xbee.devices.ZigBeeNetwork m	ethod),	method), 879	
912	g	et_file_manager()	
_dns_address()	Device	(digi.xbee.devices.WiFiDevice method), 778	
method), 768		et_file_manager()	
<del>-</del>		(digi.xbee.devices.XBeeDevice method	),
	_	=	
	ethod),		),
	<i>anager</i> g		le
	C . M	•	
	eSysteminig		
	stamManaa		
			Ŋ
	EBCCI HCSy		,
	q		
	2		!).
		565	
	g	et_firmware_version()	
(digi.xbee.devices.CellularDevice m	ethod),	(digi.xbee.devices.DigiPointDevice method	),
693		597	
	g		
(digi.xbee.devices.DigiMeshDevice m			
565	g		
			),
	_		7
	Device		),
			V
	einoa),		٠),
, -,	~		
		(digi.xbee.devices.RemoteDigiMeshDevice	
(and invocation in the property of the	· · · · · · · · · · · · · · · · · · ·	(ang invocance recontention Digities in Device	
745		method), 840	
745 _file_manager()	q	method), 840 ret_firmware_version()	
	911 _discovery_timeout()    (digi.xbee.devices.DigiMeshNetwork m 936 _discovery_timeout()    (digi.xbee.devices.DigiPointNetwork m 948 _discovery_timeout()    (digi.xbee.devices.Raw802Network m 924 _discovery_timeout()    (digi.xbee.devices.XBeeNetwork m 920 _discovery_timeout()    (digi.xbee.devices.ZigBeeNetwork m 900 _discovery_timeout()    (digi.xbee.devices.ZigBeeNetwork m 912 _dns_address() (digi.xbee.devices.WiFi method), 768 _explicit_data_received_callbac    (digi.xbee.reader.PacketListener method) _explicit_queue()    (digi.xbee.reader.PacketListener method) _explicit_queue()    (digi.xbee.filesystem.FileSystemM         method), 963 _file() (digi.xbee.filesystem.LocalXBeeFile         method), 974 _file_hash() (digi.xbee.filesystem.FileSystemEthod), 974 _file_hash() (digi.xbee.filesystem.LocalX         method), 994 _file_manager()    (digi.xbee.devices.AbstractXBeeDevice    method), 491 _file_manager()    (digi.xbee.devices.CellularDevice m 693 _file_manager()    (digi.xbee.devices.DigiMeshDevice m 565 _file_manager()    (digi.xbee.devices.DigiPointDevice m 597 _file_manager() (digi.xbee.devices.IP    method), 666 _file_manager()    (digi.xbee.devices.LPWANDevice m 719 _file_manager()	gliscovery_timeout() (digi.xbee.devices.DigiMeshNetwork method), 936 gdiscovery_timeout() (digi.xbee.devices.DigiPointNetwork method), 948 gdiscovery_timeout() (digi.xbee.devices.Raw802Network method), 924 gdiscovery_timeout() (digi.xbee.devices.XBeeNetwork method), 900 gdiscovery_timeout() (digi.xbee.devices.ZigBeeNetwork method), 912 gdns_address() (digi.xbee.devices.WiFiDevice method), 916 gexplicit_data_received_callbacks() (digi.xbee.reader.PacketListener method), 1018 gfile() (digi.xbee.filesystem.FileSystemManager gmethod), 93 gfile() (digi.xbee.filesystem.LocalXBeeFileSystemManager gmethod), 974 gfile_hash() (digi.xbee.filesystem.FileSystemManager gmethod), 974 ffile_mash() (digi.xbee.filesystem.LocalXBeeFileSystemManager gmethod), 974 ffile_manager() (digi.xbee.devices.AbstractXBeeDevice method), 693 ffile_manager() (digi.xbee.devices.CellularDevice method), 505 ffile_manager() (digi.xbee.devices.DigiMeshDevice method), 597 ffile_manager() (digi.xbee.devices.IPDevice method), 666 ffile_manager() (digi.xbee.devices.LPWANDevice method), 719 ffile_manager()	discovery_timeout() (digi.xbee.devices.DigiMeshNetwork method), 936 discovery_timeout() (digi.xbee.devices.DigiPointNetwork method), 948 discovery_timeout() (digi.xbee.devices.DigiPointNetwork method), 924 discovery_timeout() (digi.xbee.devices.Raw802Network method), 924 discovery_timeout() (digi.xbee.devices.Raw802Network method), 924 discovery_timeout() (digi.xbee.devices.Raw802Network method), 920 discovery_timeout() (digi.xbee.devices.Raw802Network method), 930 discovery_timeout() (digi.xbee.devices.Raw802Network method), 940 discovery_timeout() (digi.xbee.devices.Raw802Network method), 950 discovery_timeout() (digi.xbee.devices.Raw802Network method), 960 discovery_timeout() (digi.xbee.devices.Raw802Network method), 970 discovery_timeout() (digi.xbee.devices.Raw802Device method), 971 discovery_timeout() (digi.xbee.devices.Raw802Device method), 971 discov

method), 858	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.xbee.packets.common.ModemStatusPacket
(digi.xbee.devices.RemoteRaw802Device	method), 291
method), 820	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.xbee.packets.common.ReceivePacket
(digi.xbee.devices.RemoteXBeeDevice	method), 272
method), 803	<pre>get_frame_spec_data()</pre>
<pre>get_firmware_version()</pre>	(digi.xbee.packets.common.RemoteATCommandPacket
(digi.xbee.devices.RemoteZigBeeDevice	method), 276
method), 879	get_frame_spec_data()
get_firmware_version()	(digi.xbee.packets.common.RemoteATCommandResponsePacket method), 280
(digi.xbee.devices.WiFiDevice method), 778 get_firmware_version()	get_frame_spec_data()
(digi.xbee.devices.XBeeDevice method),	(digi.xbee.packets.common.TransmitPacket
508	method), 284
<pre>get_firmware_version()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.devices.ZigBeeDevice method),	(digi.xbee.packets.common.TransmitStatusPacket
635	method), 288
<pre>get_frame_spec_data()</pre>	<pre>get_frame_spec_data()</pre>
(digi.x bee.packets.base. Generic XBee Packet	(digi.xbee.packets.devicecloud.DeviceRequestPacket
method), 249	method), 307
<pre>get_frame_spec_data()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.packets.base.UnknownXBeePacket	(digi.x bee.packets. device cloud. Device Response Packet
method), 252	method), 310
<pre>get_frame_spec_data()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.packets.base.XBeeAPIPacket	(digi.xbee.packets.devicecloud.DeviceResponseStatusPacket
method), 246	method), 313
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.base.XBeePacket method), 245	(digi.xbee.packets.devicecloud.FrameErrorPacket method), 316
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.cellular.RXSMSPacket	(digi.xbee.packets.devicecloud.SendDataRequestPacket
method), 255	method), 320
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.cellular.TXSMSPacket	(digi.xbee.packets.devicecloud.SendDataResponsePacket
method), 258	method), 322
<pre>get_frame_spec_data()</pre>	get_frame_spec_data()
(digi.xbee.packets.common.ATCommPacket	(digi.xbee.packets.digimesh.RouteInformationPacket
method), 261	method), 327
<pre>get_frame_spec_data()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.packets.common.ATCommQueuePack	et (digi.xbee.packets.filesystem.FSRequestPacket
method), 264	method), 331
<pre>get_frame_spec_data()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.packets.common.ATCommResponsePa	acket (digi.xbee.packets.filesystem.FSResponsePacket
method), 268	method), 333
<pre>get_frame_spec_data()</pre>	<pre>get_frame_spec_data()</pre>
(digi.xbee.packets.common.ExplicitAddressingPo	
method), 298	method), 337
get_frame_spec_data()	get_frame_spec_data()
(digi.xbee.packets.common.ExplicitRXIndicator1	
<pre>method), 302 get_frame_spec_data()</pre>	<pre>method), 339 get_frame_spec_data()</pre>
	get_frame_spec_data() licatorPacketdigi.xbee.packets.network.RXIPv4Packet
method), 295	method), 344
1100110001, 270	110010001,5011

```
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.network.TXIPv4Packet
                                                             (digi.xbee.packets.socket.SocketNewIPv4ClientPacket
        method), 347
                                                            method), 421
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX16IOPacket method),
                                                             (digi.xbee.packets.socket.SocketOptionRequestPacket
                                                            method), 388
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX16Packet
                                                            (digi.xbee.packets.socket.SocketOptionResponsePacket
                                          method),
        365
                                                            method), 391
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX64IOPacket method),
                                                            (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                            method), 426
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.RX64Packet
                                                            (digi.xbee.packets.socket.SocketReceivePacket
                                          method),
                                                             method), 423
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.TX16Packet
                                          method),
                                                            (digi.xbee.packets.socket.SocketSendPacket
        354
                                                            method), 408
                                                    get_frame_spec_data()
get_frame_spec_data()
                                                            (digi.xbee.packets.socket.SocketSendToPacket
        (digi.xbee.packets.raw.TX64Packet
                                          method),
        351
                                                            method), 411
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.raw.TXStatusPacket
                                                            (digi.xbee.packets.socket.SocketStatePacket
        method), 357
                                                            method), 429
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.relay.UserDataRelayOutputPacket
                                                            (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacket
        method), 378
                                                             method), 434
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.relay.UserDataRelayPacket
                                                            (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket
        method), 375
                                                            method), 440
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.socket.SocketBindListenPacket
                                                             (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
        method), 414
                                                            method), 438
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.socket.SocketClosePacket
                                                             (digi.xbee.packets.zigbee.CreateSourceRoutePacket
        method), 401
                                                            method), 456
get_frame_spec_data()
                                                    get frame spec data()
        (digi.xbee.packets.socket.SocketCloseResponsePacket
                                                            (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
        method), 405
                                                            method), 458
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.socket.SocketConnectPacket
                                                            (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
        method), 395
                                                            method), 448
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.socket.SocketConnectResponsePacket
                                                            (digi.xbee.packets.zigbee.Register Joining Device Packet
        method), 398
                                                            method), 445
get_frame_spec_data()
                                                    get_frame_spec_data()
        (digi.xbee.packets.socket.SocketCreatePacket
                                                            (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
        method), 381
                                                            method), 452
get_frame_spec_data()
                                                    get_frame_type() (digi.xbee.packets.base.GenericXBeePacket
        (digi.xbee.packets.socket.SocketCreateResponsePacket
                                                            method), 250
        method), 385
                                                    get_frame_type() (digi.xbee.packets.base.UnknownXBeePacket
get frame spec data()
                                                            method), 252
        (digi.xbee.packets.socket.SocketListenResponsePacket_frame_type() (digi.xbee.packets.base.XBeeAPIPacket
                                                            method), 246
        method), 417
```

```
qet_frame_type() (digi.xbee.packets.cellular.RXSMSPacket_frame_type() (digi.xbee.packets.raw.RX16IOPacket
              method), 255
                                                                                                     method), 370
get_frame_type() (digi.xbee.packets.cellular.TXSMSPacket_frame_type() (digi.xbee.packets.raw.RX16Packet
                                                                                                     method), 365
              method), 258
get_frame_type() (digi.xbee.packets.common.ATCommp@acketr.ame_type() (digi.xbee.packets.raw.RX64IOPacket
              method), 261
                                                                                                     method), 368
get_frame_type() (digi.xbee.packets.common.ATComm@ue_terPawket_type() (digi.xbee.packets.raw.RX64Packet
              method), 264
                                                                                                     method), 361
get_frame_type() (digi.xbee.packets.common.ATCommgResponsaRacketpe() (digi.xbee.packets.raw.TX16Packet
              method), 268
                                                                                                     method), 355
\verb|get_frame_type()| (\textit{digi.xbee.packets.common.Explicit} \underline{\textbf{Addre}} \underline{\textbf{sxirtgRacket}} \\ \text{pe()} (\textit{digi.xbee.packets.raw.TX64Packet}) \\ \text{|get_frame_type()|} (\textit{digi.xbee.packets.ra
                                                                                                     method), 351
              method), 298
get_frame_type() (digi.xbee.packets.common.ExplicitRXIndfootbonPackete() (digi.xbee.packets.raw.TXStatusPacket
              method), 302
                                                                                                     method), 358
get_frame_type() (digi.xbee.packets.common.IOData&ganpl&RxIndicatopRackdtdigi.xbee.packets.relay.UserDataRelayOutputPack
              method), 295
                                                                                                     method), 378
get_frame_type() (digi.xbee.packets.common.ModemStatus_Rackete_type() (digi.xbee.packets.relay.UserDataRelayPacket
              method), 291
                                                                                                     method), 375
get_frame_type() (digi.xbee.packets.common.ReceiveBacketframe_type() (digi.xbee.packets.socket.SocketBindListenPacket
              method), 272
                                                                                                     method), 414
get_frame_type() (digi.xbee.packets.common.RemoteAfEConfinandPackets() (digi.xbee.packets.socket.SocketClosePacket
                                                                                                     method), 402
              method), 276
get_frame_type() (digi.xbee.packets.common.RemoteAEEContinuandResponse(Rakdigt.xbee.packets.socket.SocketCloseResponsePact
              method), 280
                                                                                                     method), 405
get_frame_type() (digi.xbee.packets.common.Transmit@ctck_dtrame_type() (digi.xbee.packets.socket.SocketConnectPacket
              method), 284
                                                                                                     method), 395
get_frame_type() (digi.xbee.packets.common.TransmitstatusfRacket_type() (digi.xbee.packets.socket.SocketConnectResponseP
              method), 288
                                                                                                     method), 399
get_frame_type() (digi.xbee.packets.devicecloud.DevigeReqfizstRacketype() (digi.xbee.packets.socket.SocketCreatePacket
                                                                                                     method), 381
              method), 308
get_frame_type() (digi.xbee.packets.devicecloud.DevigeResponsePacketpe() (digi.xbee.packets.socket.SocketCreateResponsePacketpe()
              method), 311
                                                                                                     method), 385
get_frame_type() (digi.xbee.packets.devicecloud.DevigeResponseStatusPacket(digi.xbee.packets.socket.SocketListenResponsePac
                                                                                                     method), 418
              method), 314
get_frame_type() (digi.xbee.packets.devicecloud.FrameErrofrPacket_type() (digi.xbee.packets.socket.SocketNewIPv4ClientPack
              method), 317
                                                                                                     method), 421
get_frame_type() (digi.xbee.packets.devicecloud.SendPattaRegumstPackete() (digi.xbee.packets.socket.SocketOptionRequestPack
                                                                                                     method), 388
              method), 320
get_frame_type() (digi.xbee.packets.devicecloud.SendPattaResponsePagket() (digi.xbee.packets.socket.SocketOptionResponsePa
                                                                                                     method), 392
              method), 322
get_frame_type() (digi.xbee.packets.digimesh.RouteInfartm_dtionRacketype() (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                                                                     method), 426
              method), 327
get_frame_type() (digi.xbee.packets.filesystem.FSRequestPacketme_type() (digi.xbee.packets.socket.SocketReceivePacket
              method), 331
                                                                                                     method), 423
get_frame_type() (digi.xbee.packets.filesystem.FSRespontseffackete_type() (digi.xbee.packets.socket.SocketSendPacket
              method), 334
                                                                                                     method), 408
get_frame_type() (digi.xbee.packets.filesystem.RemotefesRequestPacketpe() (digi.xbee.packets.socket.SocketSendToPacket
              method), 337
                                                                                                     method), 411
get_frame_type() (digi.xbee.packets.filesystem.RemoteFeSResponsePackete() (digi.xbee.packets.socket.SocketStatePacket
              method), 339
                                                                                                     method), 429
get_frame_type() (digi.xbee.packets.network.RXIPv4Rgæketframe_type() (digi.xbee.packets.wifi.IODataSampleRxIndicatorV
              method), 344
                                                                                                     method), 434
get_frame_type() (digi.xbee.packets.network.TXIPv4Rqxketframe_type() (digi.xbee.packets.wifi.RemoteATCommandRespon
              method), 347
                                                                                                     method), 440
```

```
get_frame_type() (digi.xbee.packets.wifi.RemoteATComputantilWifiPacketpe_value()
             method), 438
                                                                                                   (digi.xbee.packets.common.RemoteATCommandResponsePacket
get_frame_type() (digi.xbee.packets.zigbee.CreateSourceRoutePracktood), 280
              method), 456
                                                                                     get_frame_type_value()
\verb|get_frame_type|()| \textit{(digi.xbee.packets.zigbee.OTAFirmwareUpdateStatxlsPexploxdkets.common.TransmitPacket)}|
             method), 459
                                                                                                   method), 284
get_frame_type() (digi.xbee.packets.zigbee.RegisterDevite_StatusReacketpe_value()
              method), 448
                                                                                                   (digi.xbee.packets.common.TransmitStatusPacket
get_frame_type() (digi.xbee.packets.zigbee.RegisterJoiningDevinePlacellet 288
             method), 445
                                                                                     get_frame_type_value()
\verb|get_frame_type|()| \textit{ (digi.xbee.packets.zigbee. Route Record Indicat(digital bete.packets.device cloud. Device Request Packets)} \\
              method), 452
                                                                                                   method), 308
get_frame_type_value()
                                                                                     get_frame_type_value()
             (digi.xbee.packets.base.GenericXBeePacket
                                                                                                   (digi.xbee.packets.devicecloud.DeviceResponsePacket
             method), 250
                                                                                                   method), 311
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.base.UnknownXBeePacket
                                                                                                   (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket
             method), 252
                                                                                                   method), 314
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.base.XBeeAPIPacket
                                                                                                   (digi.xbee.packets.devicecloud.FrameErrorPacket
             method), 246
                                                                                                   method), 317
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.cellular.RXSMSPacket
                                                                                                   (digi.xbee.packets.devicecloud.SendDataRequestPacket
             method), 255
                                                                                                   method), 320
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.cellular.TXSMSPacket
                                                                                                   (digi.xbee.packets.devicecloud.SendDataResponsePacket
              method), 258
                                                                                                   method), 322
get_frame_type_value()
                                                                                     get_frame_type_value()
             (digi.xbee.packets.common.ATCommPacket
                                                                                                   (digi.xbee.packets.digimesh.RouteInformationPacket
             method), 262
                                                                                                   method), 327
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.common.ATCommQueuePacket
                                                                                                   (digi.xbee.packets.filesystem.FSRequestPacket
             method), 265
                                                                                                   method), 331
get_frame_type_value()
                                                                                     get_frame_type_value()
                                                                                                   (digi.xbee.packets.filesystem.FSResponsePacket
              (digi.xbee.packets.common.ATCommResponsePacket
             method), 268
                                                                                                   method), 334
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.common.ExplicitAddressingPacket
                                                                                                   (digi.xbee.packets.filesystem.RemoteFSRequestPacket
             method), 299
                                                                                                   method), 337
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.common.ExplicitRXIndicatorPacket
                                                                                                   (digi.xbee.packets.filesystem.RemoteFSResponsePacket
             method), 303
                                                                                                   method), 340
                                                                                     get_frame_type_value()
get_frame_type_value()
              (digi.xbee.packets.common. IOD at a Sample Rx Indicator Packet tidigi.xbee.packets.network. RXIPv4 Packet tidigi.xbee.packets.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.network.netw
              method), 295
                                                                                                   method), 345
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.common.ModemStatusPacket
                                                                                                   (digi.xbee.packets.network.TXIPv4Packet
             method), 291
                                                                                                   method), 348
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.common.ReceivePacket
                                                                                                   (digi.xbee.packets.raw.RX16IOPacket method),
              method), 272
                                                                                                   371
get_frame_type_value()
                                                                                     get_frame_type_value()
              (digi.xbee.packets.common.RemoteATCommandPacket
                                                                                                   (digi.xbee.packets.raw.RX16Packet
                                                                                                                                                          method),
                                                                                                   365
             method), 276
```

```
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.raw.RX64IOPacket method),
                                                            (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                            method), 426
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.raw.RX64Packet
                                         method),
                                                            (digi.xbee.packets.socket.SocketReceivePacket
                                                            method), 423
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.raw.TX16Packet
                                                            (digi.xbee.packets.socket.SocketSendPacket
                                         method),
        355
                                                            method), 408
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.raw.TX64Packet
                                         method),
                                                            (digi.xbee.packets.socket.SocketSendToPacket
                                                            method), 411
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.raw.TXStatusPacket
                                                            (digi.xbee.packets.socket.SocketStatePacket
        method), 358
                                                            method), 429
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.relay.UserDataRelayOutputPacket
                                                            (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacket
        method), 378
                                                            method), 434
                                                   get_frame_type_value()
get_frame_type_value()
        (digi.xbee.packets.relay.UserDataRelayPacket
                                                            (digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket
        method), 376
                                                            method), 440
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.socket.SocketBindListenPacket
                                                            (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
        method), 415
                                                            method), 438
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.socket.SocketClosePacket
                                                            (digi.xbee.packets.zigbee.CreateSourceRoutePacket
        method), 402
                                                            method), 456
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.socket.SocketCloseResponsePacket
                                                            (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
        method), 405
                                                            method), 459
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.socket.SocketConnectPacket
                                                            (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
        method), 396
                                                            method), 448
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.socket.SocketConnectResponsePacket
                                                            (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket
        method), 399
                                                            method), 445
get_frame_type_value()
                                                   get_frame_type_value()
        (digi.xbee.packets.socket.SocketCreatePacket
                                                            (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
        method), 382
                                                            method), 452
get_frame_type_value()
                                                   get_fs_frame_received_callbacks()
        (digi.xbee.packets.socket.SocketCreateResponsePacket
                                                            (digi.xbee.reader.PacketListener
                                                                                             method),
        method), 385
                                                            1024
                                                   get_gateway_address()
get_frame_type_value()
        (digi.xbee.packets.socket.SocketListenResponsePacket\\
                                                            (digi.xbee.devices.WiFiDevice method), 768
        method), 418
                                                   get_hardware_version()
get_frame_type_value()
                                                            (digi.xbee.devices.AbstractXBeeDevice
        (digi.xbee.packets.socket.SocketNewIPv4ClientPacket
                                                            method), 477
        method), 421
                                                   get_hardware_version()
get_frame_type_value()
                                                            (digi.xbee.devices.CellularDevice
                                                                                             method),
        (digi.xbee.packets.socket.SocketOptionRequestPacket
        method), 388
                                                   get_hardware_version()
                                                            (digi.xbee.devices.DigiMeshDevice
get frame type value()
                                                                                             method),
        (digi.xbee.packets.socket.SocketOptionResponsePacket
        method), 392
                                                   get hardware version()
```

(digi.xbee.devices.DigiPointDevice 597	method),	565 get_io_configuration()	
get_hardware_version()		(digi.xbee.devices.DigiPointDevice	method),
(digi.xbee.devices.IPDevice method),	666	597	
<pre>get_hardware_version()     (digi.xbee.devices.LPWANDevice</pre>	method),	<pre>get_io_configuration()     (digi.xbee.devices.IPDevice method)</pre>	, 667
719		<pre>get_io_configuration()</pre>	
get_hardware_version()		(digi.xbee.devices.LPWANDevice	method),
(digi.xbee.devices.NBIoTDevice	method),	719	,,,,
		get_io_configuration()	
get_hardware_version() (digi.xbee.devices.Raw802Device	method),	(digi.xbee.devices.NBIoTDevice 745	method),
532		<pre>get_io_configuration()</pre>	
get_hardware_version()		(digi.xbee.devices.Raw802Device	method),
(digi.xbee.devices.RemoteDigiMeshD method), 840	Pevice	532 get_io_configuration()	
get_hardware_version()		(digi.xbee.devices.RemoteDigiMesh.	Device
(digi.xbee.devices.RemoteDigiPointD	Pevice	method), 840	Serice
method), 859		<pre>get_io_configuration()</pre>	_
<pre>get_hardware_version()     (digi.xbee.devices.RemoteRaw802De</pre>	vice	(digi.xbee.devices.RemoteDigiPointi method), 859	Device
method), 820		<pre>get_io_configuration()</pre>	
<pre>get_hardware_version()</pre>		(digi.xbee.devices.RemoteRaw802D	evice
(digi.xbee.devices.RemoteXBeeDevice method), 803	e	<pre>method), 821 get_io_configuration()</pre>	
get_hardware_version()	ice	(digi.xbee.devices.RemoteXBeeDevices), 803	ce
method), 879		<pre>get_io_configuration()</pre>	
get_hardware_version()		(digi.xbee. devices. Remote Zig Bee Devices and the properties of the properties o	vice
(digi.xbee.devices.WiFiDevice method	d), 778	method), 880	
get_hardware_version()		<pre>get_io_configuration()</pre>	
(digi.xbee.devices.XBeeDevice	method),	(digi.xbee.devices.WiFiDevice metho	(d), 778
508		<pre>get_io_configuration()</pre>	
<pre>get_hardware_version()</pre>		(digi.xbee.devices.XBeeDevice	method),
(digi.xbee.devices.ZigBeeDevice	method),	508	
635		<pre>get_io_configuration()</pre>	
<pre>get_hsb() (digi.xbee.models.address.XBee1e     method), 201</pre>	6BitAddress		method),
get_imei_addr() (digi.xbee.devices.Cellu	ılarDevice	get_io_sample_received_callback	s ()
method), 682		(digi.xbee.reader.PacketListener met	
get_imei_addr()	'ANDevice	<pre>get_io_sampling_rate()</pre>	
method), 719		(digi.xbee.devices.AbstractXBeeDev	ice
<pre>get_imei_addr() (digi.xbee.devices.NB</pre>	IoTDevice	method), 482	
method), 745		<pre>get_io_sampling_rate()</pre>	
<pre>get_int_from_byte() (in</pre>	module	(digi.xbee.devices.CellularDevice 684	method),
get_io_configuration()		<pre>get_io_sampling_rate()</pre>	
(digi.xbee.devices.AbstractXBeeDevices)	c a	(digi.xbee.devices.DigiMeshDevice	method)
method), 482	LE	565	method),
get_io_configuration()		<pre>get_io_sampling_rate()</pre>	
(digi.xbee.devices.CellularDevice 694	method),	(digi.xbee.devices.DigiPointDevice 598	method),
get_io_configuration()		<pre>get_io_sampling_rate()</pre>	
(digi.xbee.devices.DigiMeshDevice	method),	(digi.xbee.devices.IPDevice method)	, 667

```
get_io_sampling_rate()
                                                   get_local_xbee_device()
        (digi.xbee.devices.LPWANDevice
                                         method).
                                                           (digi.xbee.devices.RemoteDigiPointDevice
                                                           method), 860
                                                   get_local_xbee_device()
get_io_sampling_rate()
        (digi.xbee.devices.NBIoTDevice
                                         method),
                                                           (digi.xbee.devices.RemoteRaw802Device
                                                           method), 822
get_io_sampling_rate()
                                                   get_local_xbee_device()
        (digi.xbee.devices.Raw802Device
                                                           (digi.xbee.devices.RemoteXBeeDevice
                                         method),
        532
                                                           method), 796
get_io_sampling_rate()
                                                   get_local_xbee_device()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                           (digi.xbee.devices.RemoteZigBeeDevice
        method), 840
                                                           method), 881
get_io_sampling_rate()
                                                   get_local_xbee_info()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                           (digi.xbee.comm_interface.XBeeCommunicationInterface
        method), 859
                                                           method), 471
get_io_sampling_rate()
                                                   get_local_xbee_info()
        (digi.xbee.devices.RemoteRaw802Device
                                                           (digi.xbee.serial.XBeeSerialPort
                                                                                            method),
        method), 821
get_io_sampling_rate()
                                                   get_lsb() (digi.xbee.models.address.XBee16BitAddress
        (digi.xbee.devices.RemoteXBeeDevice
                                                           method), 201
        method), 803
                                                   get_many_to_one_broadcasting_time()
get_io_sampling_rate()
                                                           (digi.xbee.devices.ZigBeeDevice method), 616
        (digi.xbee.devices.RemoteZigBeeDevice
                                                   get_mask_address()
        method), 880
                                                           (digi.xbee.devices.WiFiDevice method), 767
get_io_sampling_rate()
                                                   get_micropython_data_received_callbacks()
                                                           (digi.xbee.reader.PacketListener method), 1024
        (digi.xbee.devices.WiFiDevice method), 778
get_io_sampling_rate()
                                                   get_modem_status_received_callbacks()
        (digi.xbee.devices.XBeeDevice
                                                           (digi.xbee.reader.PacketListener method), 1023
                                         method),
        509
                                                   get_neighbor_table()
get_io_sampling_rate()
                                                           (digi.xbee.models.zdo.NeighborTableReader
        (digi.xbee.devices.ZigBeeDevice
                                         method),
                                                           method), 239
        636
                                                   get_neighbors() (digi.xbee.devices.DigiMeshDevice
get_ip_addr()
                    (digi.xbee.devices.CellularDevice
                                                           method), 555
        method), 694
                                                   get_neighbors()(digi.xbee.devices.RemoteDigiMeshDevice
get_ip_addr() (digi.xbee.devices.IPDevice method),
                                                           method), 833
                                                   get_neighbors() (digi.xbee.devices.RemoteZigBeeDevice
        654
get_ip_addr()
                    (digi.xbee.devices.LPWANDevice
                                                           method), 873
        method), 720
                                                   get_neighbors()
                                                                        (digi.xbee.devices.ZigBeeDevice
get_ip_addr()
                     (digi.xbee.devices.NBIoTDevice
                                                           method), 625
        method), 746
                                                   get_neighbors() (digi.xbee.models.zdo.NeighborFinder
get_ip_addr()
                       (digi.xbee.devices.WiFiDevice
                                                           method), 241
        method), 779
                                                   get_network() (digi.xbee.comm_interface.XBeeCommunicationInterfa
get_ip_addressing_mode()
                                                           method), 471
        (digi.xbee.devices.WiFiDevice method), 766
                                                                       (digi.xbee.devices.CellularDevice
                                                   get_network()
get_ip_data_received_callbacks()
                                                           method), 694
        (digi.xbee.reader.PacketListener
                                         method),
                                                                      (digi.xbee.devices.DigiMeshDevice
                                                   get_network()
        1023
                                                           method), 566
get_ip_queue()
                     (digi.xbee.reader.PacketListener
                                                                      (digi.xbee.devices.DigiPointDevice
                                                   get_network()
        method), 1018
                                                           method), 598
                                                   get_network() (digi.xbee.devices.IPDevice method),
get_local_file_hash()
                                 (in
                                          module
        digi.xbee.filesystem), 975
                                                           657
get_local_xbee_device()
                                                                       (digi.xbee.devices.LPWANDevice
                                                   get network()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                           method), 720
        method), 841
                                                   get_network()
                                                                        (digi.xbee.devices.NBIoTDevice
```

method), 746	<pre>get_node_descriptor()</pre>
<pre>get_network() (digi.xbee.devices.Raw802Device</pre>	(digi.xbee.models.zdo.NodeDescriptorReader method), 233
<pre>get_network()</pre>	<pre>get_node_id() (digi.xbee.devices.AbstractXBeeDevice</pre>
<pre>get_network()</pre>	<pre>get_node_id() (digi.xbee.devices.CellularDevice</pre>
<pre>get_network()</pre>	<pre>get_node_id() (digi.xbee.devices.DigiMeshDevice</pre>
<pre>get_network() (digi.xbee.serial.XBeeSerialPort</pre>	<pre>get_node_id() (digi.xbee.devices.DigiPointDevice</pre>
<pre>get_next_frame_id()      (digi.xbee.devices.CellularDevice method),      694</pre>	<pre>get_node_id() (digi.xbee.devices.IPDevice method),</pre>
get_next_frame_id()	method), 720
(digi.xbee.devices.DigiMeshDevice method), 566	<pre>get_node_id() (digi.xbee.devices.NBIoTDevice     method), 746</pre>
<pre>get_next_frame_id()      (digi.xbee.devices.DigiPointDevice method),</pre>	<pre>get_node_id() (digi.xbee.devices.Raw802Device</pre>
598 get_next_frame_id() (digi.xbee.devices.IPDevice	<pre>get_node_id() (digi.xbee.devices.RemoteDigiMeshDevice</pre>
<pre>method), 667 get_next_frame_id()</pre>	<pre>get_node_id() (digi.xbee.devices.RemoteDigiPointDevice</pre>
(digi.xbee.devices.LPWANDevice method), 720	<pre>get_node_id() (digi.xbee.devices.RemoteRaw802Device     method), 822</pre>
<pre>get_next_frame_id()      (digi.xbee.devices.NBIoTDevice method),</pre>	<pre>get_node_id() (digi.xbee.devices.RemoteXBeeDevice</pre>
746 get_next_frame_id()	<pre>get_node_id() (digi.xbee.devices.RemoteZigBeeDevice</pre>
(digi.xbee.devices.Raw802Device method), 533	get_node_id() (digi.xbee.devices.WiFiDevice method), 779
get_next_frame_id()	<pre>get_node_id()</pre>
(digi.xbee.devices.WiFiDevice method), 779 get_next_frame_id()	method), 509 get_node_id() (digi.xbee.devices.ZigBeeDevice
(digi.xbee.devices.XBeeDevice method),	method), 637
502 get_next_frame_id()	get_nowait() (digi.xbee.reader.XBeeQueue method), 1027
	get_nt_limits() (digi.xbee.devices.DigiMeshNetwork class method), 936
<pre>get_node_connections()      (digi.xbee.devices.DigiMeshNetwork method),      936</pre>	<pre>get_nt_limits() (digi.xbee.devices.DigiPointNetwork</pre>
get_node_connections()	class method), 924
(digi.xbee.devices.DigiPointNetwork method), 948	get_nt_limits() (digi.xbee.devices.XBeeNetwork class method), 902
get_node_connections()	<pre>get_nt_limits() (digi.xbee.devices.ZigBeeNetwork</pre>
(digi.xbee.devices.Raw802Network method),	class method), 912 get_number_devices()
<pre>get_node_connections()     (digi.xbee.devices.XBeeNetwork method),</pre>	(digi.xbee.devices.DigiMeshNetwork method), 936
904	<pre>get_number_devices()</pre>
<pre>get_node_connections()      (digi.xbee.devices.ZigBeeNetwork method),</pre>	(digi.xbee.devices.DigiPointNetwork method), 948
912	<pre>get_number_devices()</pre>

```
(digi.xbee.devices.Raw802Network
                                          method),
                                                                           (digi.xbee.devices.WiFiDevice
                                                   get_pan_id()
        924
                                                            method), 779
get_number_devices()
                                                   get_pan_id()
                                                                           (digi.xbee.devices.XBeeDevice
        (digi.xbee.devices.XBeeNetwork
                                          method),
                                                            method), 509
        894
                                                   get_pan_id()
                                                                         (digi.xbee.devices.ZigBeeDevice
get number devices()
                                                            method), 637
                                                   get_parameter() (digi.xbee.devices.AbstractXBeeDevice
        (digi.xbee.devices.ZigBeeNetwork
                                          method).
                                                            method), 473
get_ota_max_block_size()
                                                    get_parameter() (digi.xbee.devices.CellularDevice
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                            method), 694
        method), 841
                                                   \verb"get_parameter()" (\textit{digi.xbee.devices.DigiMeshDevice}")
                                                            method), 567
get_ota_max_block_size()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                   get_parameter() (digi.xbee.devices.DigiPointDevice
        method), 860
                                                            method), 599
                                                                             (digi.xbee.devices.IPDevice
get_ota_max_block_size()
                                                    get_parameter()
        (digi.xbee.devices.RemoteRaw802Device
                                                            method), 668
        method), 822
                                                    get_parameter() (digi.xbee.devices.LPWANDevice
get_ota_max_block_size()
                                                            method), 720
                                                                          (digi.xbee.devices.NBIoTDevice
        (digi.xbee.devices.RemoteXBeeDevice
                                                   get_parameter()
        method), 797
                                                            method), 747
get_ota_max_block_size()
                                                    get_parameter() (digi.xbee.devices.Raw802Device
        (digi.xbee.devices.RemoteZigBeeDevice
                                                            method), 533
        method), 881
                                                    get_parameter() (digi.xbee.devices.RemoteDigiMeshDevice
get_packet_received_callbacks()
                                                            method), 841
        (digi.xbee.reader.PacketListener
                                          method),
                                                   get_parameter() (digi.xbee.devices.RemoteDigiPointDevice
                                                            method), 860
get_packet_received_from_callbacks()
                                                    get_parameter() (digi.xbee.devices.RemoteRaw802Device
        (digi.xbee.reader.PacketListener method), 1023
                                                            method), 822
get_pan_id() (digi.xbee.devices.AbstractXBeeDevice
                                                   get_parameter() (digi.xbee.devices.RemoteXBeeDevice
        method), 480
                                                            method), 795
get_pan_id()
                    (digi.xbee.devices.CellularDevice
                                                   get_parameter() (digi.xbee.devices.RemoteZigBeeDevice
        method), 694
                                                            method), 881
get_pan_id()
                                                   get_parameter()
                   (digi.xbee.devices.DigiMeshDevice
                                                                           (digi.xbee.devices.WiFiDevice
        method), 566
                                                            method), 779
                                                   get_parameter()
get_pan_id()
                   (digi.xbee.devices.DigiPointDevice
                                                                           (digi.xbee.devices.XBeeDevice
        method), 599
                                                            method), 493
get_pan_id() (digi.xbee.devices.IPDevice method),
                                                   get_parameter()
                                                                         (digi.xbee.devices.ZigBeeDevice
        657
                                                            method), 637
                    (digi.xbee.devices.LPWANDevice
                                                   get_parameter_string()
get_pan_id()
        method), 720
                                                            (digi.xbee.models.atcomm.ATCommand
get_pan_id()
                      (digi.xbee.devices.NBIoTDevice
                                                            method), 135
        method), 746
                                                    get_phone_number_byte_array()
get_pan_id()
                                                            (digi.xbee.packets.cellular.RXSMSPacket
                    (digi.xbee.devices.Raw802Device
        method), 533
                                                            method), 254
get_pan_id() (digi.xbee.devices.RemoteDigiMeshDevice_et_phone_number_byte_array()
                                                            (digi.xbee.packets.cellular.TXSMSPacket
        method), 841
get_pan_id() (digi.xbee.devices.RemoteDigiPointDevice
                                                            method), 258
        method), 860
                                                    get_power_level()
get_pan_id() (digi.xbee.devices.RemoteRaw802Device
                                                            (digi.xbee.devices.AbstractXBeeDevice
        method), 822
                                                            method), 481
get_pan_id() (digi.xbee.devices.RemoteXBeeDevice
                                                   get_power_level()
                                                            (digi.xbee.devices.CellularDevice
        method), 804
                                                                                              method),
get_pan_id() (digi.xbee.devices.RemoteZigBeeDevice
                                                            684
        method), 881
                                                    get_power_level()
```

(digi.xbee.devices.DigiMeshDevice method), 567	<pre>get_protocol() (digi.xbee.devices.RemoteDigiMeshDevice</pre>
<pre>get_power_level()       (digi.xbee.devices.DigiPointDevice method),</pre>	<pre>get_protocol() (digi.xbee.devices.RemoteDigiPointDevice</pre>
599 get_power_level() (digi.xbee.devices.IPDevice	<pre>get_protocol() (digi.xbee.devices.RemoteRaw802Device</pre>
method), 668 get_power_level()	get_protocol() (digi.xbee.devices.RemoteXBeeDevice method), 805
(digi.xbee.devices.LPWANDevice method),	<pre>get_protocol() (digi.xbee.devices.RemoteZigBeeDevice</pre>
720 get_power_level()	method), 872 get_protocol() (digi.xbee.devices.WiFiDevice
(digi.xbee.devices.NBIoTDevice method), 747	<pre>method), 762 get_protocol() (digi.xbee.devices.XBeeDevice</pre>
<pre>get_power_level()      (digi.xbee.devices.Raw802Device method),</pre>	method), 510 get_protocol() (digi.xbee.devices.ZigBeeDevice
534	method), 616
get_power_level()	get_pwm_duty_cycle()
(digi.xbee.devices.RemoteDigiMeshDevice method), 842	(digi.xbee.devices.AbstractXBeeDevice method), 484
<pre>get_power_level()      (digi.xbee.devices.RemoteDigiPointDevice           method), 860</pre>	<pre>get_pwm_duty_cycle()     (digi.xbee.devices.CellularDevice method), 695</pre>
<pre>get_power_level()      (digi.xbee.devices.RemoteRaw802Device      method), 822</pre>	<pre>get_pwm_duty_cycle()      (digi.xbee.devices.DigiMeshDevice method), 567</pre>
get_power_level()	<pre>get_pwm_duty_cycle()</pre>
(digi.xbee.devices.RemoteXBeeDevice method), 804	(digi.xbee.devices.DigiPointDevice method), 599
<pre>get_power_level()      (digi.xbee.devices.RemoteZigBeeDevice</pre>	<pre>get_pwm_duty_cycle()     (digi.xbee.devices.IPDevice method), 668</pre>
method), 881	get_pwm_duty_cycle()
get_power_level() (digi.xbee.devices.WiFiDevice method), 780	(digi.xbee.devices.LPWANDevice method), 721
<pre>get_power_level() (digi.xbee.devices.XBeeDevice</pre>	<pre>get_pwm_duty_cycle()      (digi.xbee.devices.NBIoTDevice method),</pre>
get_power_level()	747
(digi.xbee.devices.ZigBeeDevice method), 637	<pre>get_pwm_duty_cycle()      (digi.xbee.devices.Raw802Device method),</pre>
get_protocol()	
method), 478 get_protocol() (digi.xbee.devices.CellularDevice	<pre>get_pwm_duty_cycle()     (digi.xbee.devices.RemoteDigiMeshDevice</pre>
method), 681	method), 842
<pre>get_protocol() (digi.xbee.devices.DigiMeshDevice</pre>	<pre>get_pwm_duty_cycle()     (digi.xbee.devices.RemoteDigiPointDevice)</pre>
<pre>get_protocol() (digi.xbee.devices.DigiPointDevice</pre>	<pre>method), 861 get_pwm_duty_cycle()</pre>
get_protocol() (digi.xbee.devices.IPDevice method), 668	(digi.xbee.devices.RemoteRaw802Device method), 823
<pre>get_protocol() (digi.xbee.devices.LPWANDevice</pre>	<pre>get_pwm_duty_cycle()     (digi.xbee.devices.RemoteXBeeDevice</pre>
<pre>get_protocol() (digi.xbee.devices.NBIoTDevice</pre>	<pre>method), 805 get_pwm_duty_cycle()</pre>
get_protocol() (digi.xbee.devices.Raw802Device method), 520	(digi.xbee.devices.RemoteZigBeeDevice method), 882

```
get_pwm_duty_cycle()
                                                    get_route_to_node()
        (digi.xbee.devices.WiFiDevice method), 780
                                                            (digi.xbee.devices.CellularDevice
                                                                                              method),
get_pwm_duty_cycle()
                                                            695
        (digi.xbee.devices.XBeeDevice
                                          method),
                                                   get_route_to_node()
                                                            (digi.xbee.devices.DigiMeshDevice
                                                                                              method),
get_pwm_duty_cycle()
        (digi.xbee.devices.ZigBeeDevice
                                          method).
                                                   get_route_to_node()
                                                            (digi.xbee.devices.DigiPointDevice
                                                                                              method),
                      (digi.xbee.reader.PacketListener
                                                            600
get_queue()
        method), 1018
                                                   get_route_to_node() (digi.xbee.devices.IPDevice
get_read_timeout()
                                                            method), 669
        (digi.xbee.serial.XBeeSerialPort
                                                   get_route_to_node()
                                          method),
                                                            (digi.xbee.devices.LPWANDevice
                                                                                              method),
get_role()
               (digi.xbee.devices.AbstractXBeeDevice
                                                            721
        method), 478
                                                    get_route_to_node()
get_role()
                    (digi.xbee.devices.CellularDevice
                                                            (digi.xbee.devices.NBIoTDevice
                                                                                              method),
        method), 695
                                                            747
                                                   get_route_to_node()
                   (digi.xbee.devices.DigiMeshDevice
get_role()
                                                            (digi.xbee.devices.Raw802Device
        method), 568
                                                                                              method),
get role()
                   (digi.xbee.devices.DigiPointDevice
        method), 600
                                                    get_route_to_node()
get_role() (digi.xbee.devices.IPDevice method), 669
                                                            (digi.xbee.devices.WiFiDevice method), 781
                    (digi.xbee.devices.LPWANDevice
get_role()
                                                   get_route_to_node()
        method), 721
                                                            (digi.xbee.devices.XBeeDevice
                                                                                              method),
                                                            502
get_role() (digi.xbee.devices.NBIoTDevice method),
        747
                                                    get_route_to_node()
get_role()
                    (digi.xbee.devices.Raw802Device
                                                            (digi.xbee.devices.ZigBeeDevice
                                                                                              method),
        method), 534
get_role() (digi.xbee.devices.RemoteDigiMeshDevice get_routes() (digi.xbee.devices.RemoteZigBeeDevice
        method), 842
                                                            method), 872
get_role() (digi.xbee.devices.RemoteDigiPointDevice get_routes()
                                                                         (digi.xbee.devices.ZigBeeDevice
        method), 861
                                                            method), 624
get_role() (digi.xbee.devices.RemoteRaw802Device
                                                   get_serial_port()
                                                            (digi.xbee.devices.RemoteDigiMeshDevice
        method), 823
                (digi.xbee.devices.RemoteXBeeDevice
                                                            method), 843
get role()
        method), 805
                                                   get_serial_port()
get_role()
               (digi.xbee.devices.RemoteZigBeeDevice
                                                            (digi.xbee.devices.RemoteDigiPointDevice
        method), 882
                                                            method), 861
get_role() (digi.xbee.devices.WiFiDevice method),
                                                   get_serial_port()
        780
                                                            (digi.xbee.devices.RemoteRaw802Device
get_role() (digi.xbee.devices.XBeeDevice method),
                                                            method), 823
                                                   get_serial_port()
        511
                                                            (digi.xbee.devices.RemoteXBeeDevice
get_role() (digi.xbee.devices.ZigBeeDevice method),
        638
                                                            method), 796
get_root() (digi.xbee.fileSystemManager
                                                   get_serial_port()
        method), 960
                                                            (digi.xbee.devices.RemoteZigBeeDevice
get_route_info_callbacks()
                                                            method), 882
        (digi.xbee.reader.PacketListener
                                          method),
                                                   get_setting_default_value()
                                                            (digi.xbee.profile.XBeeProfile method), 991
get_route_record_received_callbacks()
                                                    get_sms_received_callbacks()
                                                            (digi.xbee.reader.PacketListener
        (digi.xbee.reader.PacketListener method), 1024
                                                                                              method),
                                                            1023
get route table()
        (digi.xbee.models.zdo.RouteTableReader
                                                   get_sock_info() (digi.xbee.xsocket.socket method),
                                                            1038
        method), 236
```

```
get_socket_data_received_callbacks()
                                                           (digi.xbee.devices.RemoteRaw802Device
        (digi.xbee.reader.PacketListener method), 1024
                                                          method), 824
get_socket_data_received_from_callbacks()get_sync_ops_timeout()
                                                           (digi.xbee.devices.RemoteXBeeDevice
        (digi.xbee.reader.PacketListener method), 1024
get_socket_info()
                                                          method), 806
        (digi.xbee.devices.CellularDevice
                                                  get_sync_ops_timeout()
                                        method),
        683
                                                          (digi.xbee.devices.RemoteZigBeeDevice
get_socket_info()
                                                           method), 883
                                                  get_sync_ops_timeout()
        (digi.xbee.devices.LPWANDevice
                                        method),
                                                          (digi.xbee.devices.WiFiDevice method), 781
        722
get_socket_info()
                                                  get_sync_ops_timeout()
        (digi.xbee.devices.NBIoTDevice
                                                           (digi.xbee.devices.XBeeDevice
                                                                                           method),
                                        method),
                                                  get_sync_ops_timeout()
get_socket_state_received_callbacks()
        (digi.xbee.reader.PacketListener method), 1024
                                                          (digi.xbee.devices.ZigBeeDevice
                                                                                           method),
get_sockets_list()
                                                           639
        (digi.xbee.devices.CellularDevice
                                        method),
                                                  get_update_progress_callbacks()
        683
                                                          (digi.xbee.devices.DigiMeshNetwork method),
get_sockets_list()
        (digi.xbee.devices.LPWANDevice
                                        method),
                                                  get_update_progress_callbacks()
        722
                                                           (digi.xbee.devices.DigiPointNetwork method),
get_sockets_list()
                                                  get_update_progress_callbacks()
        (digi.xbee.devices.NBIoTDevice
                                        method),
                                                           (digi.xbee.devices.Raw802Network
                                                                                           method).
                                                           924
get_sync_ops_timeout()
        (digi.xbee.devices. Abstract XBee Device\\
                                                  get_update_progress_callbacks()
        method), 479
                                                           (digi.xbee.devices.XBeeNetwork
                                                                                           method),
get_sync_ops_timeout()
        (digi.xbee.devices.CellularDevice
                                        method),
                                                  get_update_progress_callbacks()
        696
                                                           (digi.xbee.devices.ZigBeeNetwork
                                                                                           method),
get_sync_ops_timeout()
                                                           912
        (digi.xbee.devices.DigiMeshDevice
                                        method),
                                                  get_usage_information()
                                                          (digi.xbee.file system. Local XBee File System Manager\\
        568
get_sync_ops_timeout()
                                                          method), 974
        (digi.xbee.devices.DigiPointDevice
                                        method),
                                                  get_user_data_relay_received_callbacks()
                                                          (digi.xbee.reader.PacketListener method), 1023
get_sync_ops_timeout()
                                                  get_volume_info()
        (digi.xbee.devices.IPDevice method), 670
                                                          (digi.xbee.filesystem.FileSystemManager
get_sync_ops_timeout()
                                                          method), 965
        (digi.xbee.devices.LPWANDevice
                                        method),
                                                  get_wifi_ai_status()
        723
                                                          (digi.xbee.devices.WiFiDevice method), 762
get_sync_ops_timeout()
                                                  get_xbee_device_callbacks()
        (digi.xbee.devices.NBIoTDevice
                                                           (digi.xbee.devices.CellularDevice
                                        method),
                                                                                           method),
        749
                                                           696
                                                  get_xbee_device_callbacks()
get_sync_ops_timeout()
        (digi.xbee.devices.Raw802Device
                                                          (digi.xbee.devices.DigiMeshDevice
                                                                                           method),
                                        method),
        535
                                                           568
get_sync_ops_timeout()
                                                  get_xbee_device_callbacks()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                          (digi.xbee.devices.DigiPointDevice
                                                                                           method),
        method), 843
get_sync_ops_timeout()
                                                  get_xbee_device_callbacks()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                          (digi.xbee.devices.IPDevice method), 670
        method), 862
                                                  get_xbee_device_callbacks()
                                                           (digi.xbee.devices.LPWANDevice
get_sync_ops_timeout()
                                                                                           method),
```

723	568
<pre>get_xbee_device_callbacks()</pre>	has_explicit_packets()
(digi.xbee.devices.NBIoTDevice method),	(digi.xbee.devices.DigiPointDevice method),
749	601
<pre>get_xbee_device_callbacks()           (digi.xbee.devices.Raw802Device method),</pre>	has_explicit_packets() (digi.xbee.devices.IPDevice method), 670
(algi.xbee.devices.Raw802Device meinoa), 535	has_explicit_packets()
get_xbee_device_callbacks()	(digi.xbee.devices.LPWANDevice method),
(digi.xbee.devices.WiFiDevice method), 781	723
<pre>get_xbee_device_callbacks()</pre>	has_explicit_packets()
(digi.xbee.devices.XBeeDevice method), 500	(digi.xbee.devices.NBIoTDevice method), 749
<pre>get_xbee_device_callbacks()</pre>	has_explicit_packets()
(digi.xbee.devices.ZigBeeDevice method), 639	(digi.xbee.devices.Raw802Device method), 535
<pre>getblocking() (digi.xbee.xsocket.socket method),</pre>	has_explicit_packets() (digi.xbee.devices.WiFiDevice method), 781
GetPathIdCmdRequest (class in	has_explicit_packets()
digi.xbee.models.filesystem), 174	$(digi.xbee.devices.XBeeDevice \qquad method),$
GetPathIdCmdResponse (class in	497
digi.xbee.models.filesystem), 176	has_explicit_packets()
<pre>getsocketopt() (digi.xbee.xsocket.socket method),</pre>	(digi.xbee.devices.ZigBeeDevice method), 639
<pre>gettimeout() (digi.xbee.xsocket.socket method),</pre>	has_filesystem (digi.xbee.profile.XBeeProfile at- tribute), 992
Н	has_firmware_files (digi.xbee.profile.XBeeProfile attribute), 992
hardware_version (digi.xbee.profile.XBeeProfile at-	has_local_filesystem
tribute), 992	$(digi.xbee.profile.XBeeProfile \ attribute),$
HardwareVersion (class in digi.xbee.models.hw),	991
192	has_local_firmware_files
has_analog_value() (digi.xbee.io.IOSample method), 984	(digi.xbee.profile.XBeeProfile attribute), 992
has_analog_values() (digi.xbee.io.IOSample method), 984	has_packets() (digi.xbee.devices.CellularDevice method), 696
has_devices() (digi.xbee.devices.DigiMeshNetwork method), 937	has_packets() (digi.xbee.devices.DigiMeshDevice method), 569
has_devices() (digi.xbee.devices.DigiPointNetwork method), 949	has_packets() (digi.xbee.devices.DigiPointDevice method), 601
	has_packets() (digi.xbee.devices.IPDevice method), 670
has_devices() (digi.xbee.devices.XBeeNetwork method), 894	has_packets() (digi.xbee.devices.LPWANDevice method), 723
has_devices() (digi.xbee.devices.ZigBeeNetwork method), 913	has_packets() (digi.xbee.devices.NBIoTDevice method), 749
has_digital_value() (digi.xbee.io.IOSample method), 983	has_packets() (digi.xbee.devices.Raw802Device method), 536
has_digital_values() (digi.xbee.io.IOSample	has_packets() (digi.xbee.devices.WiFiDevice
method), 983	method), 782
has_explicit_packets()	has_packets() (digi.xbee.devices.XBeeDevice
(digi.xbee.devices.CellularDevice method),	method), 497
696	has_packets() (digi.xbee.devices.ZigBeeDevice method), 639
<pre>has_explicit_packets()      (digi.xbee.devices.DigiMeshDevice method),</pre>	has_power_supply_value()
(digi.xbee.devices.DigiMeshDevice method),	mab_power_bappry_varac()

(digi.xbee.io.IOSample method), 984	<pre>index() (digi.xbee.reader.IPDataReceived method),</pre>
has_pwm_capability() (digi.xbee.io.IOLine	1004
method), 981	<pre>index() (digi.xbee.reader.MicroPythonDataReceived</pre>
has_remote_filesystem	method), 1007
(digi.xbee.profile.XBeeProfile attribute), 992	index() (digi.xbee.reader.ModemStatusReceived method), 999
has_remote_firmware_files	<pre>index() (digi.xbee.reader.NetworkModified method),</pre>
(digi.xbee.profile.XBeeProfile attribute), 992	1001 index() (digi.xbee.reader.NetworkUpdateProgress
HashFileCmdRequest (class in	method), 1016
digi.xbee.models.filesystem), 157	<pre>index() (digi.xbee.reader.PacketReceived method),</pre>
HashFileCmdResponse (class in	996
digi.xbee.models.filesystem), 159	index() (digi.xbee.reader.PacketReceivedFrom
hex_string_to_bytes() (in module	method), 997
digi.xbee.util.utils), 464	index() (digi.xbee.reader.RelayDataReceived
hex_to_string() (in module digi.xbee.util.utils),	method), 1006
467	index() (digi.xbee.reader.RouteInformationReceived
hops (digi.xbee.packets.zigbee.CreateSourceRoutePacket attribute), 455	method), 1012
hops (digi.xbee.packets.zigbee.RouteRecordIndicatorPack	index() (digi.xbee.reader.RouteReceived method), et 1013
attribute), 451	index() (digi.xbee.reader.RouteRecordIndicatorReceived
annonie), 451	method), 1011
	index() (digi.xbee.reader.SMSReceived method), 1005
I2C_FUNCTIONALITY (digi.xbee.io.IOMode at-	index() (digi.xbee.reader.SocketDataReceived
tribute), 985	method), 1009
id (digi.xbee.models.protocol.Role attribute), 218	<pre>index() (digi.xbee.reader.SocketDataReceivedFrom</pre>
id (digi.xbee.models.zdo.NeighborRelationship at-	method), 1010
tribute), 240	index() (digi.xbee.reader.SocketStateReceived
id (digi.xbee.models.zdo.RouteStatus attribute), 237	method), 1008
ident (digi.xbee.reader.PacketListener attribute), 1024	<pre>index() (digi.xbee.reader.XBeeEvent method), 995</pre>
index (digi.xbee.io.IOLine attribute), 980	INDIRECT_TRANSMISSION
index (digi.xbee.profile.FirmwareBaudrate attribute),	(digi.x bee.models.options. Transmit Options
986	attribute), 210
index (digi.xbee.profile.FirmwareParity attribute), 986	InitDiscoveryScan (class in digi.xbee.reader),
<pre>index (digi.xbee.profile.FirmwareStopbits attribute),</pre>	1013
987	insert() (digi.xbee.reader.BluetoothDataReceived
index() (digi.xbee.reader.BluetoothDataReceived	method), 1007
method), 1007	insert() (digi.xbee.reader.DataReceived method),
index () (digi.xbee.reader.DataReceived method), 998	998
index() (digi.xbee.reader.DeviceDiscovered method),	<pre>insert() (digi.xbee.reader.DeviceDiscovered method),</pre>
1001	insert() (digi.xbee.reader.DiscoveryProcessFinished
index() (digi.xbee.reader.DiscoveryProcessFinished method), 1002	method), 1002
index() (digi.xbee.reader.EndDiscoveryScan method),	insert() (digi.xbee.reader.EndDiscoveryScan
1014	method), 1014
index() (digi.xbee.reader.ExplicitDataReceived	<pre>insert() (digi.xbee.reader.ExplicitDataReceived</pre>
method), 1003	method), 1003
index() (digi.xbee.reader.FileSystemFrameReceived	<pre>insert() (digi.xbee.reader.FileSystemFrameReceived</pre>
method), 1015	method), 1015
<pre>index() (digi.xbee.reader.InitDiscoveryScan method),</pre>	<pre>insert() (digi.xbee.reader.InitDiscoveryScan</pre>
1014	method), 1014
<pre>index() (digi.xbee.reader.IOSampleReceived method),</pre>	insert() (digi.xbee.reader.IOSampleReceived
1000	method), 1000
1000	insert() (digi.xbee.reader.IPDataReceived method),

1004	ip_addr (digi.xbee.models.message.IPMessage at-
$\verb"insert"() \textit{ (digi.xbee.reader.MicroPythonDataReceived }$	tribute), 206
method), 1007	ip_protocol (digi.xbee.packets.network.RXIPv4Packe
insert() (digi.xbee.reader.ModemStatusReceived	attribute), 344
method), 999	ip_protocol (digi.xbee.packets.network.TXIPv4Packet
<pre>insert() (digi.xbee.reader.NetworkModified method),</pre>	attribute), 349
insert() (digi.xbee.reader.NetworkUpdateProgress	IPAddressingMode (class in digi.xbee.models.mode), 198
method), 1016	IPDataReceived (class in digi.xbee.reader), 1004
insert() (digi.xbee.reader.PacketReceived method),	IPDevice (class in digi.xbee.devices), 653
996	IPMessage (class in digi.xbee.models.message), 206
<pre>insert() (digi.xbee.reader.PacketReceivedFrom</pre>	<pre>IPProtocol (class in digi.xbee.models.protocol), 217</pre>
method), 997	$\verb is_alive()  (\textit{digi.xbee.reader.PacketListener method}),$
insert() (digi.xbee.reader.RelayDataReceived	1025
method), 1006	is_apply_changes_enabled()
<pre>insert() (digi.xbee.reader.RouteInformationReceived</pre>	(digi.xbee.devices.AbstractXBeeDevice
method), 1012	method), 479
<pre>insert() (digi.xbee.reader.RouteReceived method),</pre>	is_apply_changes_enabled() (digi.xbee.devices.CellularDevice method),
insert() (digi.xbee.reader.RouteRecordIndicatorReceive	
method), 1011	is_apply_changes_enabled()
<pre>insert() (digi.xbee.reader.SMSReceived method),</pre>	(digi.xbee.devices.DigiMeshDevice method),
1005	569
<pre>insert() (digi.xbee.reader.SocketDataReceived</pre>	is_apply_changes_enabled()
method), 1009	(digi.xbee.devices.DigiPointDevice method),
<pre>insert() (digi.xbee.reader.SocketDataReceivedFrom</pre>	601
method), 1010	is_apply_changes_enabled()
insert() (digi.xbee.reader.SocketStateReceived	(digi.xbee.devices.IPDevice method), 670
method), 1008	is_apply_changes_enabled()
<pre>insert() (digi.xbee.reader.XBeeEvent method), 995 int_to_ascii() (in module digi.xbee.util.utils), 466</pre>	(digi.xbee.devices.LPWANDevice method), 723
int_to_bytes() (in module digi.xbee.util.utils), 465	is_apply_changes_enabled()
<pre>int_to_length() (in module digi.xbee.util.utils),</pre>	(digi.xbee.devices.NBIoTDevice method),
466	749
InvalidConfigurationException, 956	is_apply_changes_enabled()
InvalidOperatingModeException, 957	(digi.xbee.devices.Raw802Device method),
<pre>InvalidPacketException, 957</pre>	536
$\verb"io_sample" (digi.x bee.packets.common.IOD at a Sample Rx) \\$	
attribute), 294	(digi.xbee.devices.RemoteDigiMeshDevice
io_sample (digi.xbee.packets.raw.RX16IOPacket at-	method), 843
tribute), 373	is_apply_changes_enabled()
io_sample (digi.xbee.packets.raw.RX64IOPacket attribute), 369	(digi.xbee.devices.RemoteDigiPointDevice method), 862
io_sample(digi.xbee.packets.wifi.IODataSampleRxIndic	
attribute), 433	(digi.xbee.devices.RemoteRaw802Device
IODataSampleRxIndicatorPacket (class in	method), 824
digi.xbee.packets.common), 292	is_apply_changes_enabled()
IODataSampleRxIndicatorWifiPacket (class	(digi.xbee.devices.RemoteXBeeDevice
in digi.xbee.packets.wifi), 431	method), 806
IOLine (class in digi.xbee.io), 980	is_apply_changes_enabled()
IOMode (class in digi.xbee.io), 985	(digi.xbee.devices.RemoteZigBeeDevice
IOSample (class in digi.xbee.io), 981	method), 883
IOSampleReceived (class in digi.xbee.reader), 999 IOValue (class in digi.xbee.io), 981	is_apply_changes_enabled() (digi.xbee.devices.WiFiDevice method), 782
$\pm \cup v$ a $\pm u \in \{ciuss in uigi.\lambda bee.io\}, 701$	(aigi.xoee.aevices.wiriDevice meimoa). / 82

method), 311

method), 314

method), 317

```
is_apply_changes_enabled()
                                                                                    is_broadcast() (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.devicecloud.SendD
              (digi.xbee.devices.XBeeDevice
                                                                    method),
                                                                                                  method), 320
                                                                                    is broadcast()(digi.xbee.packets.devicecloud.SendDataResponsePac
is_apply_changes_enabled()
                                                                                                  method), 322
              (digi.xbee.devices.ZigBeeDevice
                                                                    method),
                                                                                    is_broadcast() (digi.xbee.packets.digimesh.RouteInformationPacket
                                                                                                  method), 327
is_bit_enabled() (in module digi.xbee.util.utils), is_broadcast() (digi.xbee.packets.filesystem.FSRequestPacket
                                                                                                  method), 331
is_broadcast(digi.xbee.models.message.ExplicitXBeeMessageoadcast() (digi.xbee.packets.filesystem.FSResponsePacket
              attribute), 205
                                                                                                  method), 334
is_broadcast(digi.xbee.models.message.XBeeMessageis_broadcast()(digi.xbee.packets.filesystem.RemoteFSRequestPacket
              attribute), 204
                                                                                                  method), 338
is_broadcast() (digi.xbee.packets.base.GenericXBeePacket\tooadcast() (digi.xbee.packets.filesystem.RemoteFSResponsePacket
             method), 250
                                                                                                  method), 340
is_broadcast() (digi.xbee.packets.base.UnknownXBeePackotroadcast() (digi.xbee.packets.network.RXIPv4Packet
              method), 252
                                                                                                  method), 345
is_broadcast() (digi.xbee.packets.base.XBeeAPIPackets_broadcast() (digi.xbee.packets.network.TXIPv4Packet
                                                                                                  method), 348
             method), 247
is_broadcast() (digi.xbee.packets.cellular.RXSMSPackets_broadcast() (digi.xbee.packets.raw.RX16IOPacket
             method), 256
                                                                                                  method), 372
is_broadcast() (digi.xbee.packets.cellular.TXSMSPackets_broadcast() (digi.xbee.packets.raw.RX16Packet
                                                                                                  method), 364
             method), 259
is_broadcast() (digi.xbee.packets.common.ATCommPacketproadcast() (digi.xbee.packets.raw.RX64IOPacket
              method), 262
                                                                                                  method), 367
is_broadcast() (digi.xbee.packets.common.ATCommQiueuePacketcast() (digi.xbee.packets.raw.RX64Packet
             method), 265
                                                                                                  method), 360
is_broadcast() (digi.xbee.packets.common.ATCommResponsePacketst() (digi.xbee.packets.raw.TX16Packet
              method), 268
                                                                                                  method), 355
is_broadcast() (digi.xbee.packets.common.ExplicitAddressingPacketst() (digi.xbee.packets.raw.TX64Packet
                                                                                                  method), 352
             method), 299
is_broadcast() (digi.xbee.packets.common.ExplicitRXhrdlicotcorPackets() (digi.xbee.packets.raw.TXStatusPacket
              method), 304
                                                                                                  method), 358
is_broadcast() (digi.xbee.packets.common.IODataSaripsle_RxIndicatorsRacketdigi.xbee.packets.relay.UserDataRelayOutputPacket
                                                                                                  method), 379
              method), 293
is_broadcast() (digi.xbee.packets.common.ModemStatusPacketadcast() (digi.xbee.packets.relay.UserDataRelayPacket
             method), 291
                                                                                                  method), 376
is_broadcast() (digi.xbee.packets.common.ReceivePacket_broadcast() (digi.xbee.packets.socket.SocketBindListenPacket
                                                                                                  method), 415
             method), 270
is_broadcast() (digi.xbee.packets.common.RemoteATGcommondRinckest() (digi.xbee.packets.socket.SocketClosePacket
              method), 276
                                                                                                  method), 402
is_broadcast() (digi.xbee.packets.common.RemoteATGonnimandResportsetFoldieri.xbee.packets.socket.SocketCloseResponsePacket
              method), 280
                                                                                                  method), 405
is_broadcast() (digi.xbee.packets.common.TransmitPacketDroadcast() (digi.xbee.packets.socket.SocketConnectPacket
                                                                                                  method), 396
             method), 285
is_broadcast() (digi.xbee.packets.common.TransmitStatus_Packetdcast() (digi.xbee.packets.socket.SocketConnectResponsePack
                                                                                                  method), 399
              method), 288
is_broadcast() (digi.xbee.packets.devicecloud.DeviceResquestBacketast() (digi.xbee.packets.socket.SocketCreatePacket
             method), 308
                                                                                                  method), 382
is_broadcast() (digi.xbee.packets.devicecloud.DeviceResplanseBacketst() (digi.xbee.packets.socket.SocketCreateResponsePacket
```

1096 Index

is\_broadcast() (digi.xbee.packets.devicecloud.DeviceResplanseStatusBucket(digi.xbee.packets.socketListenResponsePacket

is\_broadcast() (digi.xbee.packets.devicecloud.FrameErrorBacketlcast() (digi.xbee.packets.socket.SocketNewIPv4ClientPacket

method), 385

*method*), 418

method), 421

```
is broadcast() (digi.xbee.packets.socket.SocketOptionRequestPacket info complete()
        method), 388
                                                           (digi.xbee.devices.IPDevice method), 654
is_broadcast() (digi.xbee.packets.socket.SocketOptionResponse:Roveketnfo_complete()
                                                           (digi.xbee.devices.LPWANDevice
        method), 392
                                                                                            method),
is_broadcast() (digi.xbee.packets.socket.SocketReceiveFromPackets
        method), 427
                                                  is device info complete()
is broadcast()(digi.xbee.packets.socket.SocketReceivePacket (digi.xbee.devices.NBIoTDevice
                                                                                            method),
        method), 423
(digi.xbee.devices.Raw802Device
        method), 408
                                                                                            method),
is_broadcast() (digi.xbee.packets.socket.SocketSendToPacket
                                                   is_device_info_complete()
        method), 412
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
is_broadcast() (digi.xbee.packets.socket.SocketStatePacket
                                                           method), 843
        method), 430
is_broadcast() (digi.xbee.packets.wifi.IODataSampleRxsln_dieaterXvijfiPacket_complete()
        method), 434
                                                           (digi.xbee.devices.RemoteDigiPointDevice
is_broadcast() (digi.xbee.packets.wifi.RemoteATCommandResponstAVAIf)Packet
        method), 441
                                                  is device info complete()
is_broadcast() (digi.xbee.packets.wifi.RemoteATCommandWifiPadkeitxbee.devices.RemoteRaw802Device
        method), 438
                                                           method), 824
is_broadcast() (digi.xbee.packets.zigbee.CreateSourceRoutePacket_info_complete()
        method), 456
                                                           (digi.xbee.devices.RemoteXBeeDevice
is_broadcast() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatethBdok&D6
                                                   is device info complete()
        method), 459
is_broadcast() (digi.xbee.packets.zigbee.RegisterDeviceStatusPudkgi.xbee.devices.RemoteZigBeeDevice
        method), 449
                                                           method), 872
is_broadcast() (digi.xbee.packets.zigbee.RegisterJoining_Devise:Packetnfo_complete()
                                                           (digi.xbee.devices.WiFiDevice method), 782
        method), 446
is_broadcast() (digi.xbee.packets.zigbee.RouteRecordInslicatorPacket nfo_complete()
        method), 451
                                                           (digi.xbee.devices.XBeeDevice
                                                                                            method),
is_connected(digi.xbee.filesystem.LocalXBeeFileSystemManager511
        attribute), 972
                                                   is_device_info_complete()
is_connected (digi.xbee.xsocket.socket attribute),
                                                           (digi.xbee.devices.ZigBeeDevice
                                                                                            method),
        1038
is connected()
                   (digi.xbee.devices.CellularDevice
                                                  is dir
                                                            (digi.xbee.filesystem.FileSystemElement
                                                           tribute), 959
        method), 682
is_connected()
                    (digi.xbee.devices.LPWANDevice
                                                  IS DIR (digi.xbee.models.options.DirResponseFlag at-
        method), 723
                                                           tribute), 216
is_connected()
                     (digi.xbee.devices.NBIoTDevice
                                                  is_discovery_running()
                                                           (digi.xbee.devices.DigiMeshNetwork method),
        method), 749
                       (digi.xbee.devices.WiFiDevice
is_connected()
        method), 765
                                                  is_discovery_running()
                                                           (digi.xbee.devices.DigiPointNetwork method),
is_device_info_complete()
        (digi.xbee.devices.AbstractXBeeDevice
                                                           949
        method), 477
                                                  is_discovery_running()
is_device_info_complete()
                                                           (digi.xbee.devices.Raw802Network
                                                                                            method),
        (digi.xbee.devices.CellularDevice
                                         method),
                                                           925
        681
                                                  is_discovery_running()
is_device_info_complete()
                                                           (digi.xbee.devices.XBeeNetwork
                                                                                            method),
        (digi.xbee.devices.DigiMeshDevice
                                         method),
        569
                                                  is_discovery_running()
is device info complete()
                                                           (digi.xbee.devices.ZigBeeNetwork
                                                                                            method),
        (digi.xbee.devices.DigiPointDevice
                                         method),
                                                           913
                                                  is_interface_open
        601
```

```
(digi.xbee.comm_interface.XBeeCommunicationInterface on () (digi.xbee.devices.XBeeDevice method),
        attribute), 470
                                                              500
is_interface_open
                                                     is open() (digi.xbee.devices.ZigBeeDevice method),
         (digi.xbee.serial.XBeeSerialPort
                                          attribute),
                                                              640
                                                     is_open() (digi.xbee.profile.XBeeProfile method), 990
                                                     is remote() (digi.xbee.devices.AbstractXBeeDevice
is known node addr()
        (digi.xbee.models.address.XBee16BitAddress
                                                              method), 479
                                                     is_remote()
        class method), 201
                                                                          (digi.xbee.devices.CellularDevice
is_known_node_addr()
                                                              method), 697
        (digi.xbee.models.address.XBee 64Bit Address
                                                     is_remote()
                                                                         (digi.xbee.devices.DigiMeshDevice
        class method), 202
                                                              method), 569
is_last(digi.xbee.models.filesystem.OpenDirCmdResponse_remote()
                                                                         (digi.xbee.devices.DigiPointDevice
        attribute), 166
                                                              method), 602
is_last(digi.xbee.models.filesystem.ReadDirCmdResponics_remote()
                                                                     (digi.xbee.devices.IPDevice method),
                                                              670
         attribute), 173
IS_LAST
            (digi.xbee.models.options.DirResponseFlag
                                                     is_remote()
                                                                          (digi.xbee.devices.LPWANDevice
        attribute), 216
                                                              method), 724
is_low_memory
                    (digi.xbee.models.zdo.Route
                                                                           (digi.xbee.devices.NBIoTDevice
                                                at-
                                                     is_remote()
        tribute), 238
                                                              method), 750
is_many_to_one
                    (digi.xbee.models.zdo.Route
                                                at-
                                                     is remote()
                                                                          (digi.xbee.devices.Raw802Device
        tribute), 238
                                                              method), 536
                                                     is_remote() (digi.xbee.devices.RemoteDigiMeshDevice
is_node_in_network()
         (digi.xbee.devices.DigiMeshNetwork method),
                                                              method), 843
                                                     is_remote() (digi.xbee.devices.RemoteDigiPointDevice
         937
is_node_in_network()
                                                              method), 862
         (digi.xbee.devices.DigiPointNetwork method),
                                                     is_remote() (digi.xbee.devices.RemoteRaw802Device
                                                              method), 824
                                                                      (digi.xbee.devices.RemoteXBeeDevice
is_node_in_network()
                                                     is_remote()
         (digi.xbee.devices.Raw802Network
                                           method),
                                                              method), 796
         925
                                                     is_remote() (digi.xbee.devices.RemoteZigBeeDevice
is_node_in_network()
                                                              method), 883
         (digi.xbee.devices.XBeeNetwork
                                           method),
                                                     is_remote() (digi.xbee.devices.WiFiDevice method),
        902
                                                              782
is_node_in_network()
                                                     is_remote() (digi.xbee.devices.XBeeDevice method),
         (digi.xbee.devices.ZigBeeNetwork
                                           method),
                                                              500
                                                                           (digi.xbee.devices.ZigBeeDevice
                                                     is remote()
is_op_mode_valid()
                                                              method), 640
        (digi.xbee.sender.PacketSender
                                                     is_route_record_required
                                           method),
         1029
                                                              (digi.xbee.models.zdo.Route attribute), 238
is_open() (digi.xbee.devices.CellularDevice method),
                                                     is_rssi (digi.xbee.devices.LinkQuality attribute), 953
                                                                           (digi.xbee.reader.PacketListener
                                                     is running()
                   (digi.xbee.devices.DigiMeshDevice
                                                              method), 1017
is_open()
                                                     is_secure (digi.xbee.fileSystemElement at-
        method), 569
                    (digi.xbee.devices.DigiPointDevice
                                                              tribute), 959
is_open()
                                                     \verb|IS_SECURE| (digi.xbee.models.options.DirResponseFlag
        method), 602
is_open() (digi.xbee.devices.IPDevice method), 670
                                                              attribute), 216
is_open() (digi.xbee.devices.LPWANDevice method),
                                                     is_valid() (digi.xbee.models.address.XBee16BitAddress
         724
                                                              class method), 201
is_open() (digi.xbee.devices.NBIoTDevice method),
                                                     is_valid() (digi.xbee.models.address.XBee64BitAddress
         750
                                                              class method), 202
is_open() (digi.xbee.devices.Raw802Device method),
                                                     is_valid() (digi.xbee.models.address.XBeeIMEIAddress
         536
                                                              class method), 203
is_open()
              (digi.xbee.devices.WiFiDevice method),
                                                     isAlive() (digi.xbee.reader.PacketListener method),
                                                              1024
         782
```

J	lq_a2b (digi.xbee.devices.Connection attribute), 954
<pre>join() (digi.xbee.reader.PacketListener method), 1025 join() (digi.xbee.reader.XBeeQueue method), 1027</pre>	lq_b2a (digi.xbee.devices.Connection attribute), 955
K	M mac_capabilities (digi.xbee.models.zdo.NodeDescriptor
key (digi.xbee.packets.zigbee.RegisterJoiningDevicePacke	
attribute), 445	make_directory() (digi.xbee.filesystem.FileSystemManager method), 961
	<pre>make_directory() (digi.xbee.filesystem.LocalXBeeFileSystemManage</pre>
LegacyHardwareVersion (class in	method), 973
digi.xbee.models.hw), 194	manufacturer_code
length (digi.xbee.packets.digimesh.RouteInformationPacattribute), 326	ket (digi.xbee.models.zdo.NodeDescriptor at- tribute), 235
<pre>length_to_int() (in module digi.xbee.util.utils),      465</pre>	<pre>max_buffer_size(digi.xbee.models.zdo.NodeDescriptor</pre>
letter (digi.xbee.models.hw.LegacyHardwareVersion	max_in_transfer_size
attribute), 195	(digi.xbee.models.zdo.NodeDescriptor at-
LinkQuality (class in digi.xbee.devices), 953	tribute), 235
<pre>list_directory() (digi.xbee.fileSystem.FileSystemMe</pre>	nmager_out_transfer_size
method), 961	(digi.xbee.models.zdo.NodeDescriptor at-
<pre>list_directory() (digi.xbee.filesystem.LocalXBeeFile</pre>	MAX_TIME_BETWEEN_REQUESTS
listen() (digi.xbee.xsocket.socket method), 1035	(digi.xbee.devices.XBeeNetwork attribute),
$\verb local_interface  (digi.x bee.models.message. User Data)   $	aRelayMessage
attribute), 208	MAX_TIME_BETWEEN_SCANS
local_port (digi.xbee.models.info.SocketInfo attribute), 196	(digi.xbee.devices.XBeeNetwork attribute), 892
LocalXBeeFileSystemManager (class in digi.xbee.filesystem), 972	MicroPythonDataReceived (class in digi.xbee.reader), 1007
log (digi.xbee.devices.AbstractXBeeDevice attribute), 491	min_io_sample_payload() (digi.xbee.io.IOSample static method), 982
log (digi.xbee.devices.CellularDevice attribute), 697	MIN_TIME_BETWEEN_REQUESTS
log (digi.xbee.devices.DigiMeshDevice attribute), 569 log (digi.xbee.devices.DigiPointDevice attribute), 602	(digi.xbee.devices.XBeeNetwork attribute), 892
log (digi.xbee.devices.IPDevice attribute), 671	MIN_TIME_BETWEEN_SCANS
log (digi.xbee.devices.LPWANDevice attribute), 724	(digi.xbee.devices.XBeeNetwork attribute),
log (digi.xbee.devices.NBIoTDevice attribute), 750	892
log (digi.xbee.devices.Raw802Device attribute), 536	modem_status(digi.xbee.packets.common.ModemStatusPacket
log (digi.xbee.devices.RemoteDigiMeshDevice at-	attribute), 290
tribute), 844	ModemStatus (class in digi.xbee.models.status), 221 ModemStatusPacket (class in
log (digi.xbee.devices.RemoteDigiPointDevice at-	digi.xbee.packets.common), 289
tribute), 862 log (digi.xbee.devices.RemoteRaw802Device attribute),	ModemStatusReceived (class in digi.xbee.reader), 998
824 log (digi.xbee.devices.RemoteXBeeDevice attribute),	move() (digi.xbee.filesystem.FileSystemManager method), 965
806 log (digi.xbee.devices.RemoteZigBeeDevice attribute),	move_element() (digi.xbee.filesystem.LocalXBeeFileSystemManager method), 973
883	memouj, 715
log (digi.xbee.devices.WiFiDevice attribute), 782 log (digi.xbee.devices.XBeeDevice attribute), 511	N
log (digi.xbee.devices.ZigBeeDevice attribute), 640	name (digi.xbee.filesystem.FileSystemElement attribute),
LPWANDevice (class in digi.xbee.devices), 708	958
1q (digi.xbee.devices.LinkQuality attribute), 953	name (digi.xbee.models.filesystem.CreateDirCmdRequest
1q (digi.xbee.models.zdo.Neighbor attribute), 241	attribute), 162

name (digi.xbee.models.filesystem.DeleteCmdRequest atneeds id() (digi.xbee.packets.common.RemoteATCommandResponsePa tribute), 183 method), 278 name (digi.xbee.models.filesystem.FileIdNameCmd atneeds id() (digi.xbee.packets.common.TransmitPacket method), 283 tribute), 142 needs\_id() (digi.xbee.packets.common.TransmitStatusPacket name (digi.xbee.models.filesystem.GetPathIdCmdRequest

method), 287

method), 310

*method*), 313

method), 316

method), 319

method), 324

method), 326

method), 330

*method*), 333

method), 336

needs\_id() (digi.xbee.packets.devicecloud.DeviceResponsePacket

needs\_id() (digi.xbee.packets.devicecloud.FrameErrorPacket

needs\_id() (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket

needs\_id() (digi.xbee.packets.devicecloud.SendDataResponsePacket

 $\verb|needs_id()| \textit{ (digi.xbee.packets.digimesh.RouteInformationPacket )}|$ 

needs\_id() (digi.xbee.packets.filesystem.RemoteFSRequestPacket

needs\_id() (digi.xbee.packets.filesystem.FSRequestPacket

needs\_id() (digi.xbee.packets.filesystem.FSResponsePacket

- name (digi.xbee.models.filesystem.HashFileCmdRequest needs id() (digi.xbee.packets.devicecloud.DeviceRequestPacket attribute), 158 method), 306
- name (digi.xbee.models.filesystem.OpenDirCmdRequest attribute), 165

attribute), 176

- name (digi.xbee.models.filesystem.OpenFileCmdRequest attribute), 144
- (digi.xbee.models.filesystem.RenameCmdRequest name attribute), 179
- name (digi.xbee.models.filesystem.VolFormatCmdRequest needs\_id() (digi.xbee.packets.devicecloud.SendDataRequestPacket attribute), 189
- (digi.xbee.models.filesystem.VolStatCmdRequest name attribute), 185
- (digi.xbee.profile.XBeeProfileSetting attribute), name
- name (digi.xbee.reader.PacketListener attribute), 1025 NBIoTDevice (class in digi.xbee.devices), 735
- ND\_PACKET\_FINISH (digi.xbee.devices.XBeeNetwork attribute), 892
- ND\_PACKET\_REMOTE (digi.xbee.devices.XBeeNetwork
- attribute), 892 needs\_id() (digi.xbee.packets.base.GenericXBeePacket needs\_id() (digi.xbee.packets.filesystem.RemoteFSResponsePacket
- *method*), 249 method), 341 needs\_id() (digi.xbee.packets.base.UnknownXBeePacketneeds\_id() (digi.xbee.packets.network.RXIPv4Packet
- *method*), 253 method), 343 needs\_id() (digi.xbee.packets.base.XBeeAPIPacket needs\_id() (digi.xbee.packets.network.TXIPv4Packet
- method), 247 method), 347 needs\_id() (digi.xbee.packets.cellular.RXSMSPacket needs\_id() (digi.xbee.packets.raw.RX16IOPacket
- *method*), 254 method), 372 needs id() (digi.xbee.packets.cellular.TXSMSPacket (digi.xbee.packets.raw.RX16Packet needs id()method), 363 method), 257
- needs id() (digi.xbee.packets.common.ATCommPacket needs id() (digi.xbee.packets.raw.RX64IOPacket method), 367 *method*), 261
- needs\_id()(digi.xbee.packets.common.ATCommQueuePackeds\_id() (digi.xbee.packets.raw.RX64Packet method), 264method), 360
- needs id()(digi.xbee.packets.common.ATCommResponsePacketid() (digi.xbee.packets.raw.TX16Packet method), 353 *method*), 267
- needs\_id() (digi.xbee.packets.common.ExplicitAddressingPacket\_id() (digi.xbee.packets.raw.TX64Packet *method*), 299 method), 350
- needs\_id() (digi.xbee.packets.common.ExplicitRXIndicatorPdcketd() (digi.xbee.packets.raw.TXStatusPacket method), 357 method), 304
- needs\_id() (digi.xbee.packets.common.IODataSampleRxIndiatoitPacketdigi.xbee.packets.relay.UserDataRelayOutputPacket method), 293 method), 377
- needs\_id() (digi.xbee.packets.common.ModemStatusPacketeds\_id() (digi.xbee.packets.relay.UserDataRelayPacket method), 290 method), 374
- needs\_id() (digi.xbee.packets.common.ReceivePacket needs\_id() (digi.xbee.packets.socket.SocketBindListenPacket *method*), 270 method), 414
- needs\_id() (digi.xbee.packets.common.RemoteATCommaradPausketd() (digi.xbee.packets.socket.SocketClosePacket method), 274 method), 401

```
needs id() (digi.xbee.packets.socket.SocketCloseResponseRawketkEventReason (class in digi.xbee.devices),
         method), 404
needs id() (digi.xbee.packets.socket.SocketConnectPackettworkEventType (class in digi.xbee.devices), 952
                                                      NetworkModified (class in digi.xbee.reader), 1000
        method), 394
needs_id() (digi.xbee.packets.socket.SocketConnectRespionxePacketPpdateProgress
        method), 398
                                                                digi.xbee.reader), 1016
needs id() (digi.xbee.packets.socket.SocketCreatePacketnew name (digi.xbee.models.filesystem.RenameCmdRequest
                                                                attribute), 179
         method), 381
needs_id() (digi.xbee.packets.socket.SocketCreateResponsePacketp (digi.xbee.models.zdo.Route attribute), 238
         method), 384
                                                      node (digi.xbee.models.zdo.Neighbor attribute), 240
needs_id() (digi.xbee.packets.socket.SocketListenResponseBacket(digi.xbee.devices.Connection attribute), 954
         method), 417
                                                       node_b (digi.xbee.devices.Connection attribute), 954
needs_id() (digi.xbee.packets.socket.SocketNewIPv4ClientPterberscriptor (class in digi.xbee.models.zdo), 234
        method), 420
                                                      NodeDescriptorReader
                                                                                           (class
needs_id() (digi.xbee.packets.socket.SocketOptionRequestPacket digi.xbee.models.zdo), 233
         method), 387
                                                      NodeUpdateType (class in digi.xbee.models.status),
needs_id() (digi.xbee.packets.socket.SocketOptionResponsePacket231
         method), 390
                                                      NONE
                                                               (digi.xbee.models.options.ReceiveOptions
needs_id() (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                               tribute), 208
        method), 428
                                                       NONE
                                                              (digi.xbee.models.options.RemoteATCmdOptions
needs_id() (digi.xbee.packets.socket.SocketReceivePacket
                                                                attribute), 211
        method), 424
                                                       NONE
                                                               (digi.xbee.models.options.TransmitOptions
needs_id() (digi.xbee.packets.socket.SocketSendPacket
                                                                tribute), 210
         method), 407
                                                      np value (digi.xbee.filesystem.FileSystemManager at-
needs id() (digi.xbee.packets.socket.SocketSendToPacket
                                                                tribute), 960
        method), 410
                                                       number_of_hops (digi.xbee.packets.zigbee.CreateSourceRoutePacket
needs_id() (digi.xbee.packets.socket.SocketStatePacket
                                                                attribute), 455
                                                      number_of_hops(digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
        method), 431
needs_id() (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiRatckidtute), 451
         method), 432
needs_id() (digi.xbee.packets.wifi.RemoteATCommandReponseWifiPacket
         method), 442
                                                       offset (digi.xbee.models.filesystem.ReadFileCmdRequest
needs_id() (digi.xbee.packets.wifi.RemoteATCommandWifiPacket attribute), 150
        method), 436
                                                      \verb|offset| (digi.x bee.models. file system. Read File CmdResponse \\
needs_id() (digi.xbee.packets.zigbee.CreateSourceRoutePacket
                                                               attribute), 152
        method), 454
                                                       offset (digi.xbee.models.filesystem.WriteFileCmdRequest
needs_id() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacketbute), 154
        method), 460
                                                                   (digi.xbee.packets.base.GenericXBeePacket
                                                       op_mode
needs_id() (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket attribute), 250
        method), 447
                                                       op_mode (digi.xbee.packets.base.UnknownXBeePacket
needs_id() (digi.xbee.packets.zigbee.RegisterJoiningDevicePacketattribute), 253
        method), 444
                                                                       (digi.xbee.packets.base.XBeeAPIPacket
                                                       op_mode
needs_id() (digi.xbee.packets.zigbee.RouteRecordIndicatorPacketattribute), 248
        method), 450
                                                       op_mode (digi.xbee.packets.base.XBeePacket attribute),
Neighbor (class in digi.xbee.models.zdo), 240
NeighborDiscoveryMode
                                     (class
                                                      op_mode (digi.xbee.packets.cellular.RXSMSPacket at-
         digi.xbee.models.mode), 199
                                                                tribute), 256
NeighborFinder (class in digi.xbee.models.zdo), 241
                                                       op_mode (digi.xbee.packets.cellular.TXSMSPacket at-
NeighborRelationship
                                    (class
                                                  in
                                                                tribute), 259
        digi.xbee.models.zdo), 239
                                                       op_mode
                                                                   (digi.xbee.packets.common.ATCommPacket
NeighborTableReader
                                   (class
                                                  in
                                                                attribute), 262
        digi.xbee.models.zdo), 238
                                                       op_mode (digi.xbee.packets.common.ATCommQueuePacket
NetworkDiscoveryStatus
                                     (class
                                                  in
                                                                attribute), 265
         digi.xbee.models.status), 227
```

- op\_mode (digi.xbee.packets.common.ATCommResponsePacket\_mode (digi.xbee.packets.raw.TX16Packet attribute), attribute), 269
- op\_mode (digi.xbee.packets.common.ExplicitAddressingPacket\_mode (digi.xbee.packets.raw.TX64Packet attribute), attribute), 299
- op\_mode(digi.xbee.packets.common.ExplicitRXIndicatorPacketnode (digi.xbee.packets.raw.TXStatusPacket atattribute), 303 tribute), 358
- op\_mode (digi.xbee.packets.common.IODataSampleRxIndioatomRadte(digi.xbee.packets.relay.UserDataRelayOutputPacket attribute), 379 attribute), 295
- op\_mode (digi.xbee.packets.common.ModemStatusPacket op\_mode (digi.xbee.packets.relay.UserDataRelayPacket attribute), 376 attribute), 291
- op\_mode (digi.xbee.packets.common.ReceivePacket at- op\_mode (digi.xbee.packets.socket.SocketBindListenPacket tribute), 272 attribute), 415
- op\_mode(digi.xbee.packets.common.RemoteATCommandRomokentode (digi.xbee.packets.socket.SocketClosePacket attribute), 402 attribute), 276
- op\_mode (digi.xbee.packets.common.RemoteATCommandRespouseBadiegi.xbee.packets.socket.SocketCloseResponsePacket attribute), 280 attribute), 405
- op\_mode (digi.xbee.packets.common.TransmitPacket at- op\_mode (digi.xbee.packets.socket.SocketConnectPacket tribute), 285 attribute), 396
- op\_mode(digi.xbee.packets.common.TransmitStatusPacketop\_mode(digi.xbee.packets.socket.SocketConnectResponsePacket attribute), 288 attribute), 399
- op\_mode (digi.xbee.packets.devicecloud.DeviceRequestPacket\_mode (digi.xbee.packets.socket.SocketCreatePacket attribute), 308 attribute), 382
- $op\_mode(digi.xbee.packets.devicecloud.DeviceResponsePacketmode(digi.xbee.packets.socket.SocketCreateResponsePacket)$ attribute), 311 attribute), 385
- op\_mode(digi.xbee.packets.devicecloud.DeviceResponseStatusRacket(digi.xbee.packets.socket.SocketListenResponsePacket attribute), 314 attribute), 418
- op\_mode (digi.xbee.packets.devicecloud.FrameErrorPacketp\_mode (digi.xbee.packets.socket.SocketNewIPv4ClientPacket attribute), 317 attribute), 421
- op\_mode (digi.xbee.packets.devicecloud.SendDataRequestBacketode (digi.xbee.packets.socket.SocketOptionRequestPacket attribute), 321 attribute), 389
- op\_mode (digi.xbee.packets.devicecloud.SendDataResponsePacketde (digi.xbee.packets.socket.SocketOptionResponsePacket attribute), 323 attribute), 392
- $\verb"op_mode" (digi.xbee.packets.digimesh.RouteInformationPacket_mode") (digi.xbee.packets.socket.SocketReceiveFromPacket_mode) (digi.xbee.packets.socket.SocketReceiveFromPacket_mode) (digi.xbee.packets.digimesh.RouteInformationPacket_mode) (digi.xbee.packets.socket.SocketReceiveFromPacket_mode) (digi.xbee.packet.SocketReceiveFromPacket_mode) (digi.xbee.packet.SocketReceiveFromPacket_mode) (digi.xbee.packet.SocketReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi.xbee.packetReceiveFromPacket_mode) (digi$ attribute), 327 attribute), 427
- op\_mode (digi.xbee.packets.filesystem.FSRequestPacket op\_mode (digi.xbee.packets.socket.SocketReceivePacket attribute), 331 attribute), 424
- op\_mode (digi.xbee.packets.filesystem.FSResponsePacket op\_mode (digi.xbee.packets.socket.SocketSendPacket atattribute), 334 tribute), 408
- op\_mode (digi.xbee.packets.filesystem.RemoteFSRequestPacketmode (digi.xbee.packets.socket.SocketSendToPacket attribute), 338 attribute), 412
- op mode (digi.xbee.packets.filesystem.RemoteFSResponsePacketode (digi.xbee.packets.socket.SocketStatePacket attribute), 340 attribute), 430
- op\_mode (digi.xbee.packets.network.RXIPv4Packet attribute), 345
- op\_mode (digi.xbee.packets.network.TXIPv4Packet attribute), 348
- op\_mode (digi.xbee.packets.raw.RX16IOPacket *tribute*), 371
- op\_mode (digi.xbee.packets.raw.RX16Packet attribute),
- (digi.xbee.packets.raw.RX64IOPacket op\_mode *tribute*), 369
- 362

- $\verb"op_mode" (\textit{digi.xbee.packets.wifi.IOD} at a \textit{SampleRxIndicatorWifiPacket}$ attribute), 435
- $op\_mode(digi.xbee.packets.wifi.RemoteATCommandResponseWifiPacket)$ attribute), 441
- op\_mode (digi.xbee.packets.wifi.RemoteATCommandWifiPacket attribute), 438
- op\_mode (digi.xbee.packets.zigbee.CreateSourceRoutePacket attribute), 457
- $op\_mode(digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket$ attribute), 459
- op\_mode (digi.xbee.packets.raw.RX64Packet attribute), op\_mode (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket attribute), 449

- $op\_mode(digi.xbee.packets.zigbee.RegisterJoiningDeviceRapketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packets.socket.SocketOptionRequestPacketon\_data(digi.xbee.packeton_dat$ attribute), 446 attribute), 387 op\_mode (digi.xbee.packets.zigbee.RouteRecordIndicatorPacketLon\_data (digi.xbee.packets.socket.SocketOptionResponsePacket
- attribute), 453 attribute), 391 open () (digi.xbee.comm\_interface.XBeeCommunicationInterface.xBeeCommunicat
- method), 470 attribute), 143
- open () (digi.xbee.devices.CellularDevice method), 681 open() (digi.xbee.devices.DigiMeshDevice method), attribute), 319

in

- open() (digi.xbee.devices.DigiPointDevice method),
- open () (digi.xbee.devices.IPDevice method), 671

550

- open () (digi.xbee.devices.LPWANDevice method), 724
- open () (digi.xbee.devices.NBIoTDevice method), 735
- open () (digi.xbee.devices.Raw802Device method), 520
- open () (digi.xbee.devices.WiFiDevice method), 762
- open () (digi.xbee.devices.XBeeDevice method), 492
- open () (digi.xbee.devices.ZigBeeDevice method), 615
- open () (digi.xbee.profile.XBeeProfile method), 990
- open () (digi.xbee.serial.XBeeSerialPort method), 1033
- OpenDirCmdRequest (class indigi.xbee.models.filesystem), 164
- OpenDirCmdResponse (class in digi.xbee.models.filesystem), 166
- OpenFileCmdRequest (class in digi.xbee.models.filesystem), 143
- OpenFileCmdResponse (class digi.xbee.models.filesystem), 145
- operating\_mode (digi.xbee.devices.CellularDevice attribute), 697
- operating\_mode (digi.xbee.devices.DigiMeshDevice attribute), 570
- operating\_mode (digi.xbee.devices.DigiPointDevice attribute), 602
- operating mode (digi.xbee.devices.IPDevice tribute), 671
- operating\_mode (digi.xbee.devices.LPWANDevice attribute), 724
- operating\_mode (digi.xbee.devices.NBIoTDevice attribute), 750
- operating\_mode (digi.xbee.devices.Raw802Device attribute), 537
- operating\_mode (digi.xbee.devices.WiFiDevice attribute), 782
- operating\_mode (digi.xbee.devices.XBeeDevice attribute), 493
- operating\_mode (digi.xbee.devices.ZigBeeDevice at*tribute*), 640
- OperatingMode (class in digi.xbee.models.mode), 197
- OperationNotSupportedException, 957
- option (digi.xbee.packets.socket.SocketOptionRequestPackettput () (digi.xbee.models.filesystem.OpenDirCmdResponse attribute), 387

- options (digi.xbee.packets.devicecloud.SendDataRequestPacket
- options (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket attribute), 444
- OPTIONS\_CLOSE\_SOCKET
  - (digi.xbee.packets.network.TXIPv4Packet attribute), 346
- OPTIONS\_LEAVE\_SOCKET\_OPEN
  - (digi.xbee.packets.network.TXIPv4Packet attribute), 346
- OTAFirmwareUpdateStatusPacket (class digi.xbee.packets.zigbee), 457
- output () (digi.xbee.models.filesystem.CloseDirCmdRequest method), 168
- output() (digi.xbee.models.filesystem.CloseDirCmdResponse method), 170
- output() (digi.xbee.models.filesystem.CloseFileCmdRequest method), 147
- output() (digi.xbee.models.filesystem.CloseFileCmdResponse method), 149
- output () (digi.xbee.models.filesystem.CreateDirCmdRequest method), 162
- output () (digi.xbee.models.filesystem.CreateDirCmdResponse method), 163
- output () (digi.xbee.models.filesystem.DeleteCmdRequest method), 183
- output() (digi.xbee.models.filesystem.DeleteCmdResponse method), 184
- (digi.xbee.models.filesystem.FileIdCmd output() method), 140
- output () (digi.xbee.models.filesystem.FileIdNameCmd method), 142
- output() (digi.xbee.models.filesystem.FSCmd method), 137
- output () (digi.xbee.models.filesystem.GetPathIdCmdRequest method), 176
- output() (digi.xbee.models.filesystem.GetPathIdCmdResponse method), 178
- output () (digi.xbee.models.filesystem.HashFileCmdRequest method), 158
- output() (digi.xbee.models.filesystem.HashFileCmdResponse *method*), 160
- output () (digi.xbee.models.filesystem.OpenDirCmdRequest method), 165
- method), 167
- option(digi.xbee.packets.socket.SocketOptionResponsePaoketput() (digi.xbee.models.filesystem.OpenFileCmdRequest attribute), 391 method), 144

output () (digi.xbee.models.filesystem.OpenFileCmdResponsteput () (digi.xbee.packets.common.ReceivePacket method), 272 method), 146 output () (digi.xbee.models.filesystem.ReadDirCmdRequestitput () (digi.xbee.packets.common.RemoteATCommandPacket method), 276 *method*), 172 output () (digi.xbee.models.filesystem.ReadDirCmdResponsePacket () (digi.xbee.packets.common.RemoteATCommandResponsePacket () method), 281 method), 174 output() (digi.xbee.models.filesystem.ReadFileCmdRequesttput() (digi.xbee.packets.common.TransmitPacket method), 151 method), 285  $\verb"output" () \textit{ (digi.xbee.models.filesystem.ReadFileCmdResp@nseput" () \textit{ (digi.xbee.packets.common.TransmitStatusPacket Packet Pack$ *method*), 153 method), 289 output () (digi.xbee.models.filesystem.RenameCmdRequestatput () (digi.xbee.packets.devicecloud.DeviceRequestPacket method), 308 *method*), 179 output () (digi.xbee.models.filesystem.RenameCmdResponsetput () (digi.xbee.packets.devicecloud.DeviceResponsePacket method), 181 method), 311 output () (digi.xbee.models.filesystem.UnknownFSCmd output () (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket method), 139 method), 314 output () (digi.xbee.models.filesystem.VolFormatCmdRequestput () (digi.xbee.packets.devicecloud.FrameErrorPacket method), 189 method), 317 output () (digi.xbee.models.filesystem.VolFormatCmdResportsput () (digi.xbee.packets.devicecloud.SendDataRequestPacket method), 191 method), 321 output () (digi.xbee.models.filesystem.VolStatCmdRequesoutput () (digi.xbee.packets.devicecloud.SendDataResponsePacket method), 323 *method*), 186 output () (digi.xbee.models.filesystem.VolStatCmdResponsetput () (digi.xbee.packets.digimesh.RouteInformationPacket method), 328 method), 188 output () (digi.xbee.models.filesystem.WriteFileCmdRequestput () (digi.xbee.packets.filesystem.FSRequestPacket method), 155 *method*), 331 output () (digi.xbee.models.filesystem.WriteFileCmdResponseput () (digi.xbee.packets.filesystem.FSResponsePacket *method*), 157 method), 334 output () (digi.xbee.packets.base.GenericXBeePacket output () (digi.xbee.packets.filesystem.RemoteFSRequestPacket method), 250 method), 338 output () (digi.xbee.packets.base.UnknownXBeePacket output () (digi.xbee.packets.filesystem.RemoteFSResponsePacket method), 253 method), 340 (digi.xbee.packets.base.XBeeAPIPacket output() (digi.xbee.packets.network.RXIPv4Packet output() method), 248method), 345 output () (digi.xbee.packets.base.XBeePacket method), (digi.xbee.packets.network.TXIPv4Packet output() method), 348 output() (digi.xbee.packets.cellular.RXSMSPacket output() (digi.xbee.packets.raw.RX16IOPacket method), 256 method), 371 (digi.xbee.packets.cellular.TXSMSPacket output () (digi.xbee.packets.raw.RX16Packet method), output() method), 259365 output() (digi.xbee.packets.common.ATCommPacket (digi.xbee.packets.raw.RX64IOPacket output() method), 262 method), 369 output () (digi.xbee.packets.common.ATCommQueuePackettput () (digi.xbee.packets.raw.RX64Packet method), *method*), 265 362 output () (digi.xbee.packets.common.ATCommResponsePackeput () (digi.xbee.packets.raw.TX16Packet method), method), 269 output () (digi.xbee.packets.common.ExplicitAddressingPacketout () (digi.xbee.packets.raw.TX64Packet method), *method*), 299 352 output () (digi.xbee.packets.common.ExplicitRXIndicatorPacketst () (digi.xbee.packets.raw.TXStatusPacket method), 303 method), 358 output () (digi.xbee.packets.common.IODataSampleRxIndiattputPacketdigi.xbee.packets.relay.UserDataRelayOutputPacket *method*), 295 method), 379

1104 Index

output () (digi.xbee.packets.common.ModemStatusPacketoutput () (digi.xbee.packets.relay.UserDataRelayPacket

method), 376

method), 291

```
output () (digi.xbee.packets.socket.SocketBindListenPacket
                                                                                                  attribute), 208
              method), 415
                                                                                    PacketListener (class in digi.xbee.reader), 1017
output() (digi.xbee.packets.socket.SocketClosePacket PacketReceived (class in digi.xbee.reader), 996
                                                                                    PacketReceivedFrom (class in digi.xbee.reader),
             method), 402
output() (digi.xbee.packets.socket.SocketCloseResponsePacket
                                                                                                  996
             method), 405
                                                                                    Packet Sender (class in digi.xbee.sender), 1028
output () (digi.xbee.packets.socket.SocketConnectPacket parameter (digi.xbee.models.atcomm.ATCommand at-
                                                                                                  tribute), 135
              method), 396
output () (digi.xbee.packets.socket.SocketConnectResponsePacket* er (digi.xbee.packets.common.ATCommPacket
              method), 399
                                                                                                  attribute), 261
output() (digi.xbee.packets.socket.SocketCreatePacket parameter(digi.xbee.packets.common.ATCommQueuePacket
                                                                                                  attribute), 264
              method), 382
\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
             method), 385
                                                                                                  attribute), 275
output () (digi.xbee.packets.socket.SocketListenResponsePacketmeter (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
              method), 418
                                                                                                  attribute), 437
output () (digi.xbee.packets.socket.SocketNewIPv4ClientPacketnt
                                                                                                    (digi.xbee.devices.RemoteZigBeeDevice at-
             method), 422
                                                                                                  tribute), 871
output () (digi.xbee.packets.socket.SocketOptionRequestPacketty
                                                                                                   (digi.xbee.profile.FirmwareParity attribute),
             method), 389
output() (digi.xbee.packets.socket.SocketOptionResponsePacket_socket_list()
             method), 392
                                                                                                  (digi.xbee.models.info.SocketInfo
                                                                                                                                                             static
output() (digi.xbee.packets.socket.SocketReceiveFromPacket
                                                                                                  method), 196
              method), 427
                                                                                    path (digi.xbee.filesystem.FileSystemElement attribute),
                                                                                                  958
output () (digi.xbee.packets.socket.SocketReceivePacket
                                                                                    path (digi.xbee.packets.devicecloud.SendDataRequestPacket
             method), 424
output() (digi.xbee.packets.socket.SocketSendPacket
                                                                                                  attribute), 319
                                                                                    PATTERN (digi.xbee.models.address.XBee16BitAddress
             method), 409
output () (digi.xbee.packets.socket.SocketSendToPacket
                                                                                                  attribute), 200
             method), 412
                                                                                    PATTERN (digi.xbee.models.address.XBee64BitAddress
output() (digi.xbee.packets.socket.SocketStatePacket
                                                                                                  attribute), 202
              method), 430
                                                                                    PATTERN (digi.xbee.models.address.XBeeIMEIAddress
output () (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacketribute), 203
                                                                                    PATTERN_PHONE_NUMBER
                                                                                                                                                           module
             method), 435
                                                                                                                                          (in
output () (digi.xbee.packets.wifi.RemoteATCommandResponseWifiRlighetbee.packets.cellular), 253
                                                                                    payload(digi.xbee.packets.socket.SocketReceiveFromPacket
             method), 441
output() (digi.xbee.packets.wifi.RemoteATCommandWifiPacket
                                                                                                  attribute), 428
              method), 438
                                                                                    payload (digi.xbee.packets.socket.SocketReceivePacket
output() (digi.xbee.packets.zigbee.CreateSourceRoutePacket
                                                                                                  attribute), 425
              method), 457
                                                                                    payload (digi.xbee.packets.socket.SocketSendPacket at-
output () (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPackeibute), 407
             method), 459
                                                                                    payload (digi.xbee.packets.socket.SocketSendToPacket
\verb"output" () \textit{ (digi.xbee.packets.zigbee.Register Device Status Packet }
                                                                                                  attribute), 411
             method), 449
                                                                                    pclose_file() (digi.xbee.filesystem.FileSystemManager
output () (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket method), 969
              method), 446
                                                                                    percent (digi.xbee.models.status.UpdateProgressStatus
output() (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
                                                                                                 attribute), 232
                                                                                    pget_file_hash() (digi.xbee.filesystem.FileSystemManager
              method), 453
                                                                                                  method), 970
Р
                                                                                    pget_path_id() (digi.xbee.fileSystem.FileSystemManager
                                                                                                  method), 966
packet (digi.xbee.sender.SyncRequestSender attribute),
                                                                                    phone_number (digi.xbee.models.message.SMSMessage
              1030
                                                                                                  attribute), 207
PACKET_ACKNOWLEDGED
                                                                                    phone number (digi.xbee.packets.cellular.RXSMSPacket
              (digi.x bee.models.options. Receive Options\\
```

attribute), 259 pilst_directory() (digi.xbee,filesystem.FileSystemManager method), 967 pmake_directory() (digi.xbee,filesystem.FileSystemManager method), 966 pop() (digi.xbee,models.options.ReceiveOptions attribute), 209 pop() (digi.xbee.reader.BuletoothDataReceived method), 969 pop() (digi.xbee.reader.DeviceDiscovered method), 989 pop() (digi.xbee.reader.DeviceDiscovered method), 1002 pop() (digi.xbee.reader.DeviceDiscovered method), 1002 pop() (digi.xbee.reader.BuletoothDataReceived method), 1002 pop() (digi.xbee.reader.DeviceDiscovered method), 1003 pop() (digi.xbee.reader.DeviceDiscovered method), 1004 pop() (digi.xbee.reader.DeviceDiscovered method), 1005 pop() (digi.xbee.reader.DeviceDiscovered method), 1005 pop() (digi.xbee.reader.DeviceDiscovered method), 1006 pop() (digi.xbee.reader.DeviceDiscovered method), 1007 pop() (digi.xbee.reader.DeviceDiscoveryScan method), 1008 pop() (digi.xbee.reader.RelaySystem.FraneReceived method), 1009 pop() (digi.xbee.reader.NetworkModified method), 1009 pop() (digi.xbee.reader.NetworkModified method), 1001 pop() (digi.xbee.reader.NetworkModified method), 1001 pop() (digi.xbee.reader.NetworkModified method), 1001 pop() (digi.xbee.reader.RelayDataReceived method), 999 pop() (digi.xbee.reader.RelayDataReceived method), 1000 pop() (digi.xbee.reader.RelayDataReceived method), 1000 pop() (digi.xbee.reader.RelayDataReceived method), 1001 pop() (digi.xbee.reader.RelayDataReceived method), 1002 pop() (digi.xbee.reader.RelayDataReceived method), 1003 pop() (digi.xbee.reader.RoutenformationReceived method), 1001 pop() (digi.xbee.reader.RoutenformationReceived method), 1001 pop() (digi.xbee.reader.RoutenformationReceived method), 1002 pop() (digi.xbee.reader.RoutenformationReceived method), 1003 pop() (digi.xbee.reader.RoutenformationReceived method), 1003 pop() (di	nhono	attribute), 254 _number (digi.xbee.packets.cellular.TXSMSPacket	pop() (digi.xbee.reader.SocketDataReceivedFrom method), 1010
coligitabee, reader Packer with method), 1007   coligitabee, reader Explicit DataReceived method), 1001   coligitabee, reader Explicit DataReceived method), 1002   coligitabee, reader Explicit DataReceived method), 1003   coligitabee, reader Explicit DataReceived method), 1004   coligitabee, reader Explicit DataReceived method), 1005   coligitabee, reader Explicit DataReceived method), 1006   coligitabee, reader Explicit DataReceived method), 1007   coligitabee, reader IND State Received method), 1008   coligitabee, reader IND State Received method), 1009   coligitabee, reader IND State Received method), 1009   coligitabee, reader IND State Received method), 1001   coligitabee, reader IND State Received method), 1004   coligitabee, reader IND State Received method), 1005   coligitabee, reader Router Individual Proper (coligitabee, reader Router Router Router Individual Proper (coligitabee, reader Router Router Individual Proper (coligitabee, rea		attribute), 259	
method), 967 method), 968 POINT_MULTIPOINT_MODE. (digi.xbee.models.options. ReceiveOptions attribute), 201 (digi.xbee.models.options. TransmitOptions attribute), 201 (digi.xbee.reader.Discovered method), 998 pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.ExplicitDataReceived method), 1005 pop () (digi.xbee.reader.ExplicitDataReceived method), 1005 pop () (digi.xbee.reader.ExplicitDataReceived method), 1001 pop () (digi.xbee.reader.ExplicitDataReceived method), 1007 pop () (digi.xbee.reader.ExplicitDataReceived method), 1007 pop () (digi.xbee.reader.PDataReceived method), 1006 pop () (digi.xbee.reader.Routefleor.Python)DataReceived method), 1007 pop () (digi.xbee.reader.Routefleor.Python)DataReceived method), 1006 pop () (digi.xbee.reader.Routefleor.Python)DataReceived method), 1007 pop () (digi.xbee.reader.Routefleor.Python)DataReceived method), 1005 pop () (digi.xbee.reader.Routefleor.Python)DataReceived method), 1005 pop () (dig	plist.	_directory()	
pmake_directory()		(digi.xbee.filesystem.FileSystemManager	pop() (digi.xbee.reader.XBeeEvent method), 995
(digi.xbee.rader.Bribesystem.BribesystemManager method), 096 POINT_MULTIPOINT_MODE (digi.xbee.rader.Bribute), 209 POINT_MULTIPOINT_MODE (digi.xbee.rader.Bribute), 209 POINT_MULTIPOINT_MODE (digi.xbee.rader.Bribute), 201 (digi.xbee.rader.Bribute), 201 (digi.xbee.rader.Bribute), 201 (digi.xbee.rader.Bribute), 209 POINT (digi.xbee.rader.Bribute), 201 (digi.xbee.rader.Bribute), 201 (digi.xbee.rader.Bribute), 201 (digi.xbee.rader.Bribute), 202 POPO ( digi.xbee.rader.Bribute), 203 POPO ( digi.xbee.rader.Bribute), 203 POPO ( digi.xbee.rader.Bribute), 204 POPO ( digi.xbee.rader.Bribute), 205 POPO ( digi.xbee.rader.Bribute), 206 POPO ( digi.xbee.rader.Bribute), 207 POPO ( digi.xbee.rader.Bribute), 207 POPO ( digi.xbee.rader.Bribute), 207 POPO ( digi.xbee.rader.Bribute), 207 POPO ( digi.		method), 967	<pre>popen_file() (digi.xbee.filesystem.FileSystemManager</pre>
method), 966 POINT_MULTIPOINT_MODE (digi.xbee.models.options.ReceiveOptions attribute), 209 POINT_MULTIPOINT_MODB (digi.xbee.models.options.TransmitOptions attribute), 211 pop () (digi.xbee.models.options.TransmitOptions attribute), 211 pop () (digi.xbee.reader.BluetoothDataReceived method), 1007 pop () (digi.xbee.reader.DutaReceived method), 998 pop () (digi.xbee.reader.DutaReceived method), 1002 pop () (digi.xbee.reader.DutaReceived method), 1003 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1016 pop () (digi.xbee.reader.BluetoothDataReceived method), 1017 pop () (digi.xbee.reader.BluetoothDataReceived method), 1016 pop () (digi.xbee.reader.BluetoothDataReceived method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1017 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1010 pop () (digi.xbee.reader.BluetoothDataReceived method), 1007 pop () (digi.xbee.reader.BluetoothDataReceived method), 1007 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1007 pop () (digi.xbee.reader.FileSystemFrameRe	pmake.	_directory()	method), 968
method), 966 POINT_MULTIPOINT_MODE (digi.xbee.models.options.ReceiveOptions attribute), 209 POINT_MULTIPOINT_MODB (digi.xbee.models.options.TransmitOptions attribute), 211 pop () (digi.xbee.models.options.TransmitOptions attribute), 211 pop () (digi.xbee.reader.BluetoothDataReceived method), 1007 pop () (digi.xbee.reader.DutaReceived method), 998 pop () (digi.xbee.reader.DutaReceived method), 1002 pop () (digi.xbee.reader.DutaReceived method), 1003 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1016 pop () (digi.xbee.reader.BluetoothDataReceived method), 1017 pop () (digi.xbee.reader.BluetoothDataReceived method), 1016 pop () (digi.xbee.reader.BluetoothDataReceived method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1017 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1010 pop () (digi.xbee.reader.BluetoothDataReceived method), 1007 pop () (digi.xbee.reader.BluetoothDataReceived method), 1007 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1007 pop () (digi.xbee.reader.FileSystemFrameRe		(digi.xbee.filesystem.FileSystemManager	<pre>power_supply_value</pre>
POINT_MULTIFOINT_MODE  (digixbee.models.options.ReceiveOptions attribute), 209  POINT_MULTIFOINT_MODE  (digixbee.models.options.TransmitOptions attribute), 211  pop () (digixbee.reader.BluetoothDataReceived method), 1007  pop () (digixbee.reader.Discovered method), 1002  pop () (digixbee.reader.DiscoveryProcessFinished method), 1007  pop () (digixbee.reader.DiscoveryScan method), 1015  pop () (digixbee.reader.BluetoothDataReceived method), 1015  pop () (digixbee.reader.BluetoothDataReceived method), 1010  pop () (digixbee.reader.BluetoothDataReceived method), 1015  pop () (digixbee.reader.BluetoothDataReceived method), 1016  pop () (digixbee.reader.BluetoothDataReceived method), 1010  pop () (digixbee.reader.BluetoothDataReceived method), 1010  pop () (digixbee.reader.DiscoveryProcessFinished method), 1015  pop () (digixbee.reader.BluetoothDataReceived method), 1010  pop () (digixbee.reader.BluetoothDataReceived method), 1010  pop () (digixbee.reader.BluetoothDataReceived method), 1010  pop () (digixbee.reader.DiscoveryProcessFinished method), 1010  pop () (digixbee.reader.DiscoveryProcessFinished method), 1010  pop () (digixbee.reader.BluetoothDataReceived method), 1015  pop () (digixbee.reader.DiscoveryProcessFinished method), 1010  pop () (digixbee.reader.BluetoothDataReceived method), 1010  pop () (digixbee.reader.DiscoveryProcessFinished method), 1010  pop () (digixbee.reader.Bluelnoothod), 1016  pop () (digixbee.reader.BluelnoothOptions method), 1016  pop () (digixbee.reader.NetworkModified method), 1016  pop () (digixbee.reader.Received method), 1017  pop () (digixbee.reader.Received method), 1018  pop () (digixbee.reader.Recei			
(digi.xbee.models.options.ReceiveOptions attribute), 209  POINT_MOTITIPOINT_MODE (digi.xbee.models.options.TransmitOptions attribute), 211  pop () (digi.xbee.reader.BluetoothDataReceived method), 1007  pop () (digi.xbee.reader.DataReceived method), 1002  pop () (digi.xbee.reader.DataReceived method), 1002  pop () (digi.xbee.reader.DataReceived method), 1002  pop () (digi.xbee.reader.DataReceived method), 1003  pop () (digi.xbee.reader.EndDiscoveryProcessFinished method), 1005  pop () (digi.xbee.reader.EndDiscoveryScan method), 1015  pop () (digi.xbee.reader.ExplicitDataReceived method), 1014  pop () (digi.xbee.reader.ExplicitDataReceived method), 1014  pop () (digi.xbee.reader.BileSystemFrameReceived method), 1004  pop () (digi.xbee.reader.BileSystemFrameReceived method), 1006  pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007  pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007  pop () (digi.xbee.reader.NetworkModified method), 1007  pop () (digi.xbee.reader.Received method), 1006  pop () (digi.xbee.reader.Received method), 1007  pop () (digi.xbee.reader.Received method), 1007  pop () (digi.xbee.reader.Received method), 1006  pop () (digi.xbee.reader.Received method), 1007  pop () (digi.xbee.reader.Received method), 1008  pop () (digi.xbee.reader.Received method), 1009  pop () (digi.xbee.reader.Received method), 1006  pop () (digi.xbee.reader.Received method), 1006  pop () (digi.xbee.reader.Received method), 1007  pop () (digi.xbee.reader.Received method), 1008	POINT		
attribute), 209  POINT_MULTIPOINT_MODE  (digi.xbee.models.options.TransmitOptions attribute), 211  pop () (digi.xbee.reader.BluetoothDataReceived method), 1007  pop () (digi.xbee.reader.DeviceDiscovered method), 1009  pop () (digi.xbee.reader.DeviceDiscovered method), 1000  pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002  pop () (digi.xbee.reader.EndDiscoveryScan method), 1015  pop () (digi.xbee.reader.EndDiscoveryScan method), 1015  pop () (digi.xbee.reader.EndDiscoveryScan method), 1016  pop () (digi.xbee.reader.EndDiscoveryScan method), 1019  pop () (digi.xbee.reader.DeviceDiscoveryScan method), 10104  pop () (digi.xbee.reader.DeviceDiscoveryScan method), 1009  pop () (digi.xbee.reader.DeviceDiscoveryScan method), 1001  pop () (digi.xbee.reader.NetworkScan method), 1000  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.RelayDataReceived method), 1001  pop () (digi.xbee.reader.RelayDataReceived method), 1002  pop () (digi.xbee.reader.RelayDataReceived method), 1006  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.SocketDataReceived method), 1013  pop () (digi.xbee.reader.SocketDataReceived method), 1015  pop () (digi.xbee.reader.RouteReceived meth			
POINT_MULTIPOINT_MODE			
(digi.xbee.models.options.TransmitOptions attribute), 211  pop () (digi.xbee.reader.BluetoothDataReceived method), 1007  pop () (digi.xbee.reader.DataReceived method), 998 pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.BnliDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1016 pop () (digi.xbee.reader.EndDiscoveryScan method), 1016 pop () (digi.xbee.reader.EndDiscoveryScan method), 1014 pop () (digi.xbee.reader.BnliDiscoveryScan method), 1014 pop () (digi.xbee.reader.InitDiscoveryScan method), 1004 pop () (digi.xbee.reader.InitDiscoveryScan meth	POINT		
attribute), 211 premove () (digi.xbee.reader.DataReceived method), 1007 pop () (digi.xbee.reader.DataReceived method), 1002 pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.ExplicitDataReceived method), 1015 pop () (digi.xbee.reader.DiscoveryScan method), 1015 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1015 pop () (digi.xbee.reader.DiscoveryScan method), 1016 pop () (digi.xbee.reader.MolemStatusReceived method), 1000 pop () (digi.xbee.reader.MolemStatusReceived method), 1007 pop () (digi.xbee.reader.MolemStatusReceived method), 1007 pop () (digi.xbee.reader.NetvorkModified method), 1001 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.RelayDataReceived method), 1012 pop () (digi.xbee.reader.RelayDataReceived method), 1012 pop () (digi.xbee.reader.RouteInformationReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SocketDataReceived method), 1015 pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.SocketDataReceived method), 1025 pop () (digi.xbee.reader.SocketDataRece	- 0 - 1 1 - 1		
pop () (digi.xbee.reader.DataReceived method), 1007 pop () (digi.xbee.reader.DataReceived method), 998 pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1016 pop () (digi.xbee.reader.InitDiscoveryScan method), 1016 pop () (digi.xbee.reader.InitDiscoveryScan method), 1000 pop () (digi.xbee.reader.InitDiscoveryScan method), 1004 pop () (digi.xbee.reader.Nowlendor) pop () (digi.xbee.reader.Nowlendor) pop () (digi.xbee.reader.Nowlendor) pop () (digi.xbee.reader.Nowlendor) pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1005 pop () (digi.xbee.reader.RouteReceived m			
method), 1007 pop () (digi.xbee.reader.DataReceived method), 998 pop () (digi.xbee.reader.Dotscovery method), 1002 pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1016 pop () (digi.xbee.reader.InitDiscoveryScan method), 1000 pop () (digi.xbee.reader.InitDiscoveryScan method), 1001 pop () (digi.xbee.reader.InitDiscoveryScan method), 1001 pop () (digi.xbee.reader.IDSampleReceived method), 1000 pop () (digi.xbee.reader.ModemStatusReceived method), 1001 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.RelayDataReceived method), 1016 pop () (digi.xbee.reader.RelayDataReceived method), 1012 pop () (digi.xbee.reader.RouteInformationReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.SocketDataReceived method), 1005 pop () (digi.xbee.reader.RouteReceived meth	non ()		
pop () (digi.xbee.reader.DataReceived method), 998 pop () (digi.xbee.reader.Discoveryd method), 1002 pop () (digi.xbee.reader.DiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.ExplicitDataReceived method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1016 pop () (digi.xbee.reader.InitDiscoveryScan method), 1010 pop () (digi.xbee.reader.InitDiscoveryScan method), 1000 pop () (digi.xbee.reader.InitDiscoveryScan method), 1000 pop () (digi.xbee.reader.InitDiscoveryScan method), 1000 pop () (digi.xbee.reader.IPDataReceived method), 1000 pop () (digi.xbee.reader.IPDataReceived method), 1000 pop () (digi.xbee.reader.ModemStatusReceived method), 1001 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.Reader.Received method), 1001 pop () (digi.xbee.reader.Reader.Received method), 1006 pop () (digi.xbee.reader.RelayDataReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.RouteReceived method), 1016 pop () (digi.xbee.reader.RouteReceived method), 1017 pop () (digi.xbee.reader.RouteReceived method), 1018 pop () (digi.xbee.reader.RouteReceived method), 1019 pop () (digi.xbee.reader.RouteReceived method), 1010 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.RouteReceived method), 1016 pop () (digi.xbee.reader.RouteReceived method), 1017 pop () (digi.xbee.reader.RouteReceived method), 1018 pop () (digi.xbee.reader.RouteReceived method), 1019 pop () (digi.xbee.reader.RouteRec	pop()		
pop () (digi.xbee.reader.DiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.EndDiscoveryScan method), 1016 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1011 pop () (digi.xbee.reader.IntiDiscoveryScan method), 1000 pop () (digi.xbee.reader.IntiDiscoveryScan method), 1001 pop () (digi.xbee.reader.IntiDiscoveryScan method), 1000 pop () (digi.xbee.reader.IntiDiscoveryScan method), 1001 pop () (digi.xbee.reader.IntiDiscoveryScan method), 1001 pop () (digi.xbee.reader.IntiDiscoveryScan method), 1004 pop () (digi.xbee.reader.IntiDiscoveryScan method), 1006 pop () (digi.xbee.reader.IntiDiscoverySca	( )		
pop () (digi.xbee.reader.EndDiscoveryProcessFinished method), 1002 pop () (digi.xbee.reader.EndDiscoveryScan method), 1013 pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1015 pop () (digi.xbee.reader.InitDiscoveryScan method), 1014 pop () (digi.xbee.reader.InitDiscoveryScan method), 1000 pop () (digi.xbee.reader.IOSampleReceived method), 1004 pop () (digi.xbee.reader.IOSampleReceived method), 1004 pop () (digi.xbee.reader.IPDataReceived method), 1004 pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007 pop () (digi.xbee.reader.ModemStatusReceived method), 1007 pop () (digi.xbee.reader.NetworkModified method), 1016 pop () (digi.xbee.reader.Received method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.RelayDataReceived method), 1016 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.RouteReceived method), 1016 pop () (digi.xbee.reader.RouteReceived method), 1017 pop () (digi.xbee.reader.RouteReceived method), 1018 pop () (digi.xbee.reader.RouteReceived method), 1019 pop () (digi.xbee.reader.RouteReceived method), 1010 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.RouteReceived method), 1016 pop () (digi.xbee.reader.RouteReceived method), 1017 pop () (digi.xbee.reader.RouteReceived method), 1018 pop () (digi.xbee.reader.RouteReceived method), 1019 pop () (digi.xbee.reader.RouteReceived method), 1010 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.RouteReceived method), 1016 pop () (digi.xbee.reader.RouteReceived method), 1017 pop () (digi.xbee.reader.RouteReceived method), 1018 pop () (digi.xbee.reader.RouteReceived method), 1019 pop () (dig			
(digi.xbee.reader.DiscoveryProcessFinished method), 1002	pop()		
method), 1002  pop () (digi.xbee.reader.EndDiscoveryScan method), 1015  pop () (digi.xbee.reader.ExplicitDataReceived method), 1015  pop () (digi.xbee.reader.FileSystemFrameReceived method), 1015  pop () (digi.xbee.reader.InitDiscoveryScan method), 1015  pop () (digi.xbee.reader.InitDiscoveryScan method), 1016  pop () (digi.xbee.reader.IPDataReceived method), 1004  pop () (digi.xbee.reader.IPDataReceived method), 1004  pop () (digi.xbee.reader.ModemStatusReceived method), 1007  pop () (digi.xbee.reader.MotworkUndateProgress method), 1016  pop () (digi.xbee.reader.PacketReceived method), 1016  pop () (digi.xbee.reader.Received method), 1017  pop () (digi.xbee.reader.RouteReceived method), 1018  pop () (digi.xbee.reader.RouteReceived method), 1019  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1015  pop () (di			
pop () (digi.xbee.reader.EndDiscoveryScan method), 1015 pop () (digi.xbee.reader.ExplicitDataReceived method), 1015 pop () (digi.xbee.reader.FileSystemFrameReceived method), 1015 pop () (digi.xbee.reader.InitDiscoveryScan method), 1014 pop () (digi.xbee.reader.InitDiscoveryScan method), 1004 pop () (digi.xbee.reader.IOSampleReceived method), 1004 pop () (digi.xbee.reader.IPDataReceived method), 1004 pop () (digi.xbee.reader.IPDataReceived method), 1007 pop () (digi.xbee.reader.ModemStatusReceived method), 1001 pop () (digi.xbee.reader.NetworkIDdateProgress method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1006 pop () (digi.xbee.reader.PacketReceived method), 1006 pop () (digi.xbee.reader.PacketReceived method), 1006 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SocketDataReceived method), 1015 pop () (digi.xbee.reader.SocketDataReceived method), 1005 pop () (digi.xbee.reader.SocketData	pop()		
attribute), 976  pop () (digi.xbee.reader.ExplicitDataReceived method), 1015  pop () (digi.xbee.reader.FileSystemFrameReceived method), 1014  pop () (digi.xbee.reader.InitDiscoveryScan method), 1014  pop () (digi.xbee.reader.IOSampleReceived method), 1000  pop () (digi.xbee.reader.IDDataReceived method), 1000  pop () (digi.xbee.reader.IDDataReceived method), 1000  pop () (digi.xbee.reader.MotorDythonDataReceived method), 1007  pop () (digi.xbee.reader.NetworkDydateProgress method), 1016  pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016  pop () (digi.xbee.reader.PacketReceived method), 997  pop () (digi.xbee.reader.PacketReceived method), 1006  pop () (digi.xbee.reader.RelayDataReceived method), 1006  pop () (digi.xbee.reader.RelayDataReceived method), 1012  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.SMSReceived method), 1005  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.SMSReceived method), 1015  pop () (digi.xbee.reader.SMSReceived method), 1015  pop () (digi.xbee.reader.RouteReceived method), 1015  pop () (digi.xbee.r			
pop() (digi.xbee.reader.ExplicitDataReceived method), 1015 pop() (digi.xbee.reader.InitDiscoveryScan method), 1014 pop() (digi.xbee.reader.InitDiscoveryScan method), 1014 pop() (digi.xbee.reader.InitDiscoveryScan method), 1000 pop() (digi.xbee.reader.IOSampleReceived method), 1000 pop() (digi.xbee.reader.NetworkDodateReceived method), 1000 pop() (digi.xbee.reader.NetworkDodateProgress method), 1016 pop() (digi.xbee.reader.RelayDataReceived method), 1000 pop() (digi.xbee.reader.RelayDataReceived method), 1012 pop() (digi.xbee.reader.RouteReceived method), 1012 pop() (digi.xbee.reader.RouteReceived method), 1011 pop() (digi.xbee.reader.RouteReceived method), 1011 pop() (digi.xbee.reader.RosteReceived method), 1015 pop() (digi.xbee.reader.RosteReceived method), 1011 pop() (digi.xbee.reader.RosteReceived method), 1015 pop() (digi.xbee.reader.RosteReceived method), 1011 pop() (digi.xbee.reader.RosteReceived method), 1015 pop() (digi.xbee.reader.RosteReceived method), 1016 pop() (digi.xbee.reader.RosteReceived method), 1011 pop() (digi.xbee.reader.RosteReceived method), 1011 pop() (digi.xbee.reader.RosteReceived method), 1012 pop() (digi.xbee.reader.RosteReceived method), 1015 pop() (digi.xbee.reader.RosteReceived method), 1005 pop() (di	pop()	•	
1003   (digi.xbee.reader.FileSystemFrameReceived method), 1015   pop () (digi.xbee.reader.InitDiscoveryScan method), 1014   pop () (digi.xbee.reader.IOSampleReceived method), 1000   pop () (digi.xbee.reader.IPDataReceived method), 1007   pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007   pop () (digi.xbee.reader.NetworkModified method), 1001   pop () (digi.xbee.reader.NetworkModified method), 1005   pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016   pop () (digi.xbee.reader.RelayDataReceived method), 1006   pop () (digi.xbee.reader.RelayDataReceived method), 1007   pop () (digi.xbee.reader.RouteReceived method), 1012   pop () (digi.xbee.reader.RouteReceived method), 1013   pop () (digi.xbee.reader.RouteReceived method), 1011   pop () (digi.xbee.reader.RouteReceived method), 1005   pop () (digi.xbe		1015	attribute), 976
pop () (digi.xbee.reader.FileSystemFrameReceived method), 1015 pop () (digi.xbee.reader.InitDiscoveryScan method), 1014 pop () (digi.xbee.reader.IOSampleReceived method), 1000 pop () (digi.xbee.reader.IPDataReceived method), 1004 pop () (digi.xbee.reader.ModemStatusReceived method), 1007 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkModified method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1006 pop () (digi.xbee.reader.PacketReceived method), 1006 pop () (digi.xbee.reader.RelayDataReceived method), 1012 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.	pop()	(digi.xbee.reader.ExplicitDataReceived method),	profile_description_file
method), 1015 pop () (digi.xbee.reader.InitDiscoveryScan method), 1014 pop () (digi.xbee.reader.IOSampleReceived method), 1000 pop () (digi.xbee.reader.IPDataReceived method), 1004 pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007 pop () (digi.xbee.reader.ModemStatusReceived method), 1007 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkWpdateProgress method), 1016 pop () (digi.xbee.reader.PacketReceived method), 996 pop () (digi.xbee.reader.PacketReceived method), 997 pop () (digi.xbee.reader.RelayDataReceived method), 1006 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.SouteReceived method), 1005 pop () (digi.xbee.reader.RouteReceived method), 1005 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.RouteReceived method), 1005 pop () (digi.xbee.read		1003	(digi.xbee.profile.XBeeProfile attribute),
pop() (digi.xbee.reader.IntiDiscoveryScan method), 1014 pop() (digi.xbee.reader.IOSampleReceived method), 1004 pop() (digi.xbee.reader.IPDataReceived method), 1004 pop() (digi.xbee.reader.MicroPythonDataReceived method), 1007 pop() (digi.xbee.reader.ModemStatusReceived method), 1007 pop() (digi.xbee.reader.NetworkModified method), 1001 pop() (digi.xbee.reader.NetworkModified method), 1001 pop() (digi.xbee.reader.NetworkUpdateProgress method), 1016 pop() (digi.xbee.reader.PacketReceived method), 997 pop() (digi.xbee.reader.PacketReceived method), 1016 pop() (digi.xbee.reader.RelayDataReceived method), 1018 pop() (digi.xbee.reader.RouteReceived method), 1012 pop() (digi.xbee.reader.RouteReceived method), 1013 pop() (digi.xbee.reader.RouteReceived method), 1011 pop() (digi.xbee.reader.SMSReceived method), 1005 pop() (digi.xbee.reader.SMS	pop()	(digi.xbee.reader.FileSystemFrameReceived	993
pop () (digi.xbee.reader.IOSampleReceived method), 1000  pop () (digi.xbee.reader.IPDataReceived method), 1004  pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007  pop () (digi.xbee.reader.ModemStatusReceived method), 1007  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016  pop () (digi.xbee.reader.PacketReceived method), 1016  pop () (digi.xbee.reader.PacketReceived method), 1016  pop () (digi.xbee.reader.RelayDataReceived method), 1006  pop () (digi.xbee.reader.RelayDataReceived method), 1012  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.SocketDataReceived method), 1005		method), 1015	<pre>profile_file (digi.xbee.profile.XBeeProfile at-</pre>
pop () (digi.xbee.reader.IOSampleReceived method), 1000  pop () (digi.xbee.reader.IPDataReceived method), 1004  pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007  pop () (digi.xbee.reader.ModemStatusReceived method), 1007  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016  pop () (digi.xbee.reader.PacketReceived method), 1016  pop () (digi.xbee.reader.PacketReceived method), 1016  pop () (digi.xbee.reader.RelayDataReceived method), 1006  pop () (digi.xbee.reader.RelayDataReceived method), 1012  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.SocketDataReceived method), 1005	pop()	(digi.xbee.reader.InitDiscoveryScan method),	tribute), 991
pop () (digi.xbee.reader.IOSampleReceived method), 1000 pop () (digi.xbee.reader.BDataReceived method), 1004 pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007 pop () (digi.xbee.reader.ModemStatusReceived method), 1007 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.PacketReceived method), 1016 pop () (digi.xbee.reader.RelayDataReceived method), 1017 pop () (digi.xbee.reader.RouteReceived method), 1018 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005 pop () (digi.		-	profile id(digi.xbee.models.message.ExplicitXBeeMessage
pop () (digi.xbee.reader.IPDataReceived method), 1004 pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007 pop () (digi.xbee.reader.ModemStatusReceived method), 1007 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016 pop () (digi.xbee.reader.PacketReceived method), 996 pop () (digi.xbee.reader.PacketReceived method), 997 pop () (digi.xbee.reader.PacketReceived method), 997 pop () (digi.xbee.reader.PacketReceived method), 1012 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SMSReceived method),	() gog	(digi.xbee.reader.IOSampleReceived method).	
pop () (digi.xbee.reader.IPDataReceived method), 1004  pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007  pop () (digi.xbee.reader.ModemStatusReceived method), pop () (digi.xbee.reader.NetworkModified method), pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016  pop () (digi.xbee.reader.PacketReceived method), pop () (digi.xbee.reader.PacketReceived method), pop () (digi.xbee.reader.PacketReceived method), pop () (digi.xbee.reader.RelayDataReceived method), pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.SocketDataReceived method), 1005	1 1 17		
pop () (digi.xbee.reader.MicroPythonDataReceived method), 1007 pop () (digi.xbee.reader.ModemStatusReceived method), 999 pop () (digi.xbee.reader.NetworkModified method), 1001 pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016 pop () (digi.xbee.reader.PacketReceived method), 996 pop () (digi.xbee.reader.PacketReceived method), 997 pop () (digi.xbee.reader.RelayDataReceived method), 1006 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1005 pop () (digi.xbee.reader.RouteReceived method), 1005 pop () (digi.xbee.reader.RouteReceived method), 1005 pop () (digi.xbee.reader.RouteReceived method), 1001 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SMSReceived met	() gog		
method), 1007  pop () (digi.xbee.reader.ModemStatusReceived method), 999  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016  pop () (digi.xbee.reader.PacketReceived method), 996  pop () (digi.xbee.reader.PacketReceived from method), 997  pop () (digi.xbee.reader.RelayDataReceived method), 996  pop () (digi.xbee.reader.RelayDataReceived method), 1016  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.RouteReceived method), 1015  pop () (digi.xbee.reader.RouteReceived method), 1005  pop () (digi.xbee.reader.SocketDataReceived method), 1005			
pop () (digi.xbee.reader.ModemStatusReceived method), 999  pop () (digi.xbee.reader.NetworkModified method), 1001  pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016  pop () (digi.xbee.reader.PacketReceived method), 996  pop () (digi.xbee.reader.PacketReceived method), 996  pop () (digi.xbee.reader.RelayDataReceived method), 997  pop () (digi.xbee.reader.RelayDataReceived method), 1012  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1015  pop () (digi.xbee.reader.SocketDataReceived method), 1005	Pop()		
pop () (digi.xbee.reader.NetworkModified method), 1016 profile_settings (digi.xbee.profile.XBeeProfile attribute), 992 pop () (digi.xbee.reader.PacketReceived method), 1016 profile_settings (digi.xbee.firmware.UpdateConfigurer method), 1026 profile_set.	non()		
pop() (digi.xbee.reader.NetworkUpdateProgress method), 1016 progress_cb() (digi.xbee.reader.PacketReceived method), 996 propopopopopopopopopopopopopopopopopopo	Pob()		
tribute), 992  pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016 progress_cb() (digi.xbee.firmware.UpdateConfigurer method), 977  pop () (digi.xbee.reader.PacketReceived method), 996 prop () (digi.xbee.reader.RelayDataReceived method), 1006 protocol (digi.xbee.models.info.SocketInfo attribute), 196  pop () (digi.xbee.reader.RouteInformationReceived method), 1012 prop () (digi.xbee.reader.RouteReceived method), 1013 protocol (digi.xbee.profile.XBeeProfile attribute), 993 progress_cb() (digi.xbee.profile.XBeeProfile attribute), 994 protocol (digi.xbee.profile.XBeeProfile attribute), 995 protocol (digi.xbee.profile.XBeeProfile attribute), 996 protocol (digi.xbee.profile.XBeeProfile attribute), 1011 method), 1012 prop () (digi.xbee.reader.SMSReceived method), 1005 prop () (digi.xbee.reader.SocketDataReceived method), 1005 prop () (digi.xbee.reader.SocketDataReceived method), 1005 prop () (digi.xbee.reader.SocketDataReceived method), 1005 protocol (digi.xbee.filesystem.FileSystemManager)	()		
pop () (digi.xbee.reader.NetworkUpdateProgress method), 1016 pop () (digi.xbee.reader.PacketReceived method), 996 pop () (digi.xbee.reader.PacketReceivedFrom method), 997 pop () (digi.xbee.reader.RelayDataReceived method), 1016 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005	pop()		
method), 1016  pop () (digi.xbee.reader.PacketReceived method), 996  pop () (digi.xbee.reader.PacketReceivedFrom method), 997  pop () (digi.xbee.reader.RelayDataReceived method), 997  pop () (digi.xbee.reader.RelayDataReceived method), 1006  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.SMSReceived method), 1005  pop () (digi.xbee.reader.SocketDataReceived method), 1005			· · · · · · · · · · · · · · · · · · ·
pop () (digi.xbee.reader.PacketReceived method), 996 pop () (digi.xbee.reader.PacketReceivedFrom method), 997 pop () (digi.xbee.reader.RelayDataReceived method), 1006 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005	pop()		
pop () (digi.xbee.reader.PacketReceivedFrom method), 997  pop () (digi.xbee.reader.RelayDataReceived method), 1006  pop () (digi.xbee.reader.RouteInformationReceived method), 1012  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1013  pop () (digi.xbee.reader.RouteReceived method), 1011  pop () (digi.xbee.reader.SMSReceived method), 1005  pop () (digi.xbee.reader.SMSReceived method), 1005  pop () (digi.xbee.reader.SocketDataReceived method), 1005  pop () (digi.xbee.reader.SocketDataReceived method), 1005  put_dir () (digi.xbee.filesystem.FileSystemManager			
997 pop () (digi.xbee.reader.RelayDataReceived method), 1006 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.RouteReceived method), 1015 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005 put_dir() (digi.xbee.filesystem.FileSystemManager	pop()		
pop () (digi.xbee.reader.RelayDataReceived method), 1006 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005 put_dir() (digi.xbee.filesystem.FileSystemManager	pop()		
1006 tribute), 206 pop () (digi.xbee.reader.RouteInformationReceived method), 1012 attribute), 381 pop () (digi.xbee.reader.RouteReceived method), 1013 protocol (digi.xbee.profile.XBeeProfile attribute), 993 pop () (digi.xbee.reader.RouteRecordIndicatorReceived method), 1011 method), 1011 pop () (digi.xbee.reader.SMSReceived method), 1005 put () (digi.xbee.reader.XBeeQueue method), 1027 pop () (digi.xbee.reader.SocketDataReceived method), put_dir() (digi.xbee.filesystem.FileSystemManager			
pop () (digi.xbee.reader.RouteInformationReceived method), 1012 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1013 pop () (digi.xbee.reader.RouteReceived method), 1011 pop () (digi.xbee.reader.SMSReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005 pop () (digi.xbee.reader.SocketDataReceived method), 1005 put () (digi.xbee.filesystem.FileSystemManager	pop()	(digi.xbee.reader.RelayDataReceived method),	
method), 1012  pop() (digi.xbee.reader.RouteReceived method), 1013  pop() (digi.xbee.reader.RouteReceived method), 1013  pop() (digi.xbee.reader.RouteRecordIndicatorReceived method), 1011  pop() (digi.xbee.reader.SMSReceived method), 1005  pop() (digi.xbee.reader.SocketDataReceived method), 1005  pop() (digi.xbee.reader.SocketDataReceived method), 1005  put() (digi.xbee.filesystem.FileSystemManager		1006	tribute), 206
pop() (digi.xbee.reader.RouteReceived method), 1013 protocol (digi.xbee.profile.XBeeProfile attribute), 993 pop() (digi.xbee.reader.RouteRecordIndicatorReceived method), 1011 method), 1032 pop() (digi.xbee.reader.SMSReceived method), 1005 put() (digi.xbee.reader.XBeeQueue method), 1027 pop() (digi.xbee.reader.SocketDataReceived method), put_dir() (digi.xbee.filesystem.FileSystemManager	pop()	(digi.xbee.reader.RouteInformationReceived	<pre>protocol (digi.xbee.packets.socket.SocketCreatePacket</pre>
pop() (digi.xbee.reader.RouteRecordIndicatorReceived purge_port() (digi.xbee.serial.XBeeSerialPort method), 1011 method), 1032 pop() (digi.xbee.reader.SMSReceived method), 1005 put() (digi.xbee.reader.XBeeQueue method), 1027 pop() (digi.xbee.reader.SocketDataReceived method), put_dir() (digi.xbee.filesystem.FileSystemManager		method), 1012	attribute), 381
pop() (digi.xbee.reader.RouteRecordIndicatorReceived purge_port() (digi.xbee.serial.XBeeSerialPort method), 1011 method), 1032 pop() (digi.xbee.reader.SMSReceived method), 1005 put() (digi.xbee.reader.XBeeQueue method), 1027 pop() (digi.xbee.reader.SocketDataReceived method), put_dir() (digi.xbee.filesystem.FileSystemManager	pop()	(digi.xbee.reader.RouteReceived method), 1013	protocol (digi.xbee.profile.XBeeProfile attribute), 993
method), 1011 method), 1032 pop() (digi.xbee.reader.SMSReceived method), 1005 put() (digi.xbee.reader.XBeeQueue method), 1027 pop() (digi.xbee.reader.SocketDataReceived method), put_dir() (digi.xbee.filesystem.FileSystemManager			
pop() (digi.xbee.reader.SMSReceived method), 1005 put() (digi.xbee.reader.XBeeQueue method), 1027 pop() (digi.xbee.reader.SocketDataReceived method), put_dir() (digi.xbee.filesystem.FileSystemManager	'		
pop() (digi.xbee.reader.SocketDataReceived method), put_dir() (digi.xbee.filesystem.FileSystemManager	() gog		
	T - T- ()		method), 964

```
put_dir() (digi.xbee.filesystem.LocalXBeeFileSystemMaRageD (digi.xbee.models.options.FileOpenRequestOption
         method), 974
                                                               attribute), 215
put_file() (digi.xbee.filesystem.FileSystemManager READ_AS_MANY (digi.xbee.models.filesystem.ReadFileCmdRequest
                                                               attribute), 150
         method), 963
put_file() (digi.xbee.filesystem.LocalXBeeFileSystemManager_byte()
                                                                            (digi.xbee.serial.XBeeSerialPort
                                                               method), 1031
        method), 973
put_nowait() (digi.xbee.reader.XBeeQueue method),
                                                                            (digi.xbee.serial.XBeeSerialPort
                                                      read bytes()
         1027
                                                               method), 1031
                                                                           (digi.xbee.devices.CellularDevice
PWM (digi.xbee.io.IOMode attribute), 985
                                                      read_data()
                                                               method), 697
pwm_at_command (digi.xbee.io.IOLine attribute), 980
pwrite_file() (digi.xbee.filesystem.FileSystemManageread_data()
                                                                          (digi.xbee.devices.DigiMeshDevice
        method), 970
                                                               method), 570
                                                                          (digi.xbee.devices.DigiPointDevice
                                                      read_data()
Q
                                                               method), 602
                                                      read_data()
                                                                      (digi.xbee.devices.IPDevice method),
qsize() (digi.xbee.reader.XBeeQueue method), 1027
quit_reading() (digi.xbee.comm_interface.XBeeCommunicationInterface
                                                                            (digi.xbee.devices.LPWANDevice
                                                      read_data()
         method), 471
                                                               method), 724
quit_reading()
                      (digi.xbee.serial.XBeeSerialPort
                                                                             (digi.xbee.devices.NBIoTDevice
         method), 1031
                                                      read_data()
                                                               method), 750
R
                                                      read_data()
                                                                           (digi.xbee.devices.Raw802Device
                                                               method), 537
Raw802Device (class in digi.xbee.devices), 519
                                                      read_data() (digi.xbee.devices.WiFiDevice method),
Raw802Network (class in digi.xbee.devices), 916
                                                               783
reachable (digi.xbee.devices.AbstractXBeeDevice at-
                                                      read_data() (digi.xbee.devices.XBeeDevice method),
         tribute), 491
                                                               496
reachable
               (digi.xbee.devices.CellularDevice
                                                 at-
                                                                            (digi.xbee.devices.ZigBeeDevice
                                                      read_data()
         tribute), 697
                                                               method), 640
              (digi.xbee.devices.DigiMeshDevice
reachable
                                                 at-
                                                      read_data_from() (digi.xbee.devices.CellularDevice
        tribute), 570
                                                               method), 697
reachable
              (digi.xbee.devices.DigiPointDevice
                                                 at-
                                                      read_data_from() (digi.xbee.devices.DigiMeshDevice
        tribute), 602
                                                               method), 570
reachable (digi.xbee.devices.IPDevice attribute), 671
                                                      read_data_from() (digi.xbee.devices.DigiPointDevice
               (digi.xbee.devices.LPWANDevice
reachable
                                                 at-
                                                               method), 603
         tribute), 724
                                                      read_data_from()
                                                                                 (digi.xbee.devices.IPDevice
reachable (digi.xbee.devices.NBIoTDevice attribute),
                                                               method), 658
         750
                                                      read_data_from() (digi.xbee.devices.LPWANDevice
reachable
               (digi.xbee.devices.Raw802Device
                                                 at-
                                                               method), 724
        tribute), 537
                                                      read_data_from() (digi.xbee.devices.NBIoTDevice
reachable (digi.xbee.devices.RemoteDigiMeshDevice
                                                               method), 750
         attribute), 844
                                                      read data from() (digi.xbee.devices.Raw802Device
reachable (digi.xbee.devices.RemoteDigiPointDevice
                                                               method), 537
         attribute), 862
                                                      read_data_from()
                                                                              (digi.xbee.devices.WiFiDevice
reachable
              (digi.xbee.devices.RemoteRaw802Device
                                                               method), 783
         attribute), 824
                                                      read_data_from()
                                                                              (digi.xbee.devices.XBeeDevice
reachable (digi.xbee.devices.RemoteXBeeDevice at-
                                                               method), 496
        tribute), 806
                                                      \verb"read_data_from"() \textit{ (digi.xbee.devices.ZigBeeDevice)}
reachable (digi.xbee.devices.RemoteZigBeeDevice at-
                                                               method), 641
        tribute), 883
                                                      read_device_info()
reachable (digi.xbee.devices.WiFiDevice attribute),
                                                               (digi.xbee.devices.AbstractXBeeDevice
         782
                                                               method), 476
reachable (digi.xbee.devices.XBeeDevice attribute),
                                                      read device info()
         511
                                                               (digi.xbee.devices.CellularDevice
                                                                                                  method),
reachable (digi.xbee.devices.ZigBeeDevice attribute),
                                                               697
```

640

read_device_info()	<pre>read_expl_data_from()</pre>
(digi.xbee.devices.DigiMeshDevice method), 571	(digi.xbee.devices.ZigBeeDevice method), 619
read_device_info()	<pre>read_file() (digi.xbee.filesystem.FileSystemManager</pre>
(digi.xbee.devices.DigiPointDevice method),	method), 962
603	<pre>read_io_sample() (digi.xbee.devices.AbstractXBeeDevice</pre>
<pre>read_device_info() (digi.xbee.devices.IPDevice</pre>	method), 483
method), 671	<pre>read_io_sample() (digi.xbee.devices.CellularDevice</pre>
<pre>read_device_info()</pre>	method), 698
(digi.xbee.devices.LPWANDevice method), 724	<pre>read_io_sample() (digi.xbee.devices.DigiMeshDevice</pre>
<pre>read_device_info()</pre>	read_io_sample() (digi.xbee.devices.DigiPointDevice
(digi.xbee.devices.NBIoTDevice method),	method), 604
750	read_io_sample() (digi.xbee.devices.IPDevice
read_device_info()	method), 672
(digi.xbee.devices.Raw802Device method),	read_io_sample() (digi.xbee.devices.LPWANDevice
538	method), 725
read_device_info()	read_io_sample() (digi.xbee.devices.NBIoTDevice
(digi.xbee.devices.RemoteDigiMeshDevice	method), 751
method), 844	read_io_sample() (digi.xbee.devices.Raw802Device
read_device_info()	method), 538
(digi.xbee.devices.RemoteDigiPointDevice	read_io_sample() (digi.xbee.devices.RemoteDigiMeshDevic
method), 863	method), 844
<pre>read_device_info()</pre>	read_io_sample() (digi.xbee.devices.RemoteDigiPointDevice
(digi.xbee.devices.RemoteRaw802Device	method), 863
method), 824	<pre>read_io_sample() (digi.xbee.devices.RemoteRaw802Device</pre>
<pre>read_device_info()</pre>	method), 825
$(digi.xbee.devices.RemoteXBeeDevice) \label{fig:control}$	read_io_sample() (digi.xbee.devices.RemoteXBeeDevice
method), 806	method), 807
read_device_info()	<pre>read_io_sample() (digi.xbee.devices.RemoteZigBeeDevice</pre>
(digi.xbee.devices.RemoteZigBeeDevice	method), 884
method), 883	read_io_sample() (digi.xbee.devices.WiFiDevice
<pre>read_device_info()</pre>	method), 783
(digi.xbee.devices.WiFiDevice method), 783	<pre>read_io_sample() (digi.xbee.devices.XBeeDevice</pre>
read_device_info()	method), 512
(digi.xbee.devices.XBeeDevice method),	read_io_sample() (digi.xbee.devices.ZigBeeDevice
512	method), 642
<pre>read_device_info()</pre>	read_ip_data() (digi.xbee.devices.CellularDevice
(digi.xbee.devices.ZigBeeDevice method),	method), 698
642	read_ip_data() (digi.xbee.devices.IPDevice
read_existing() (digi.xbee.serial.XBeeSerialPort	method), 656
method), 1031	read_ip_data() (digi.xbee.devices.LPWANDevice
read_expl_data() (digi.xbee.devices.DigiMeshDevic	- · · · · · · · · · · · · · · · · · · ·
method), 552	read_ip_data() (digi.xbee.devices.NBIoTDevice
	, _ ,
read_expl_data() (digi.xbee.devices.DigiPointDevic	
method), 585	read_ip_data() (digi.xbee.devices.WiFiDevice
read_expl_data() (digi.xbee.devices.ZigBeeDevice	method), 783
method), 618	read_ip_data_from()
read_expl_data_from()	(digi.xbee.devices.CellularDevice method),
(digi.xbee.devices.DigiMeshDevice method),	698
552	read_ip_data_from() (digi.xbee.devices.IPDevice
read_expl_data_from()	method), 657
(digi.xbee.devices.DigiPointDevice method),	read_ip_data_from()
585	(digi.xbee.devices.LPWANDevice method),

726	(22
726	622
read_ip_data_from()  (digitable devices NRIoTDevice method)	register_joining_device_async() (digi.xbee.devices.ZigBeeDevice method),
(digi.xbee.devices.NBIoTDevice method), 752	(digi.xbee.devices.ZigBeeDevice method), 623
read_ip_data_from()	RegisterDeviceStatusPacket (class in
(digi.xbee.devices.WiFiDevice method), 784	digi.xbee.packets.zigbee), 446
ReadDirCmdRequest (class in digi.xbee.models.filesystem), 171	RegisterJoiningDevicePacket (class in digi.xbee.packets.zigbee), 443
	RegisterKeyOptions (class in
digi.xbee.models.filesystem), 172	digi.xbee.models.options), 214
	registrant_address
digi.xbee.models.filesystem), 150	(digi.xbee.packets.zigbee.RegisterJoiningDevicePacket
ReadFileCmdResponse (class in	attribute), 444
digi.xbee.models.filesystem), 152	relationship (digi.xbee.models.zdo.Neighbor
ReadProfileException, 989	attribute), 240
real_status (digi.xbee.packets.common.ATCommResp attribute), 267	onserayneatakecelved (class in algi.xbee.reader), 1005
real_status(digi.xbee.packets.common.RemoteATCom	
attribute), 279	attribute), 196
receive_options (digi.xbee.packets.common.Explicit. attribute), 305	RXI IndiantorPacketsss (digi.xbee.packets.socket.SocketNewIPv4ClientPacket attribute), 420
	SampleRxI <u>n</u> dientocPadigi.xbee.models.message.ExplicitXBeeMessage
attribute), 294	attribute), 205
receive_options (digi.xbee.packets.common.Receive.attribute), 271	Packerote_device (digi.xbee.models.message.XBeeMessage attribute), 204
receive_options (digi.xbee.packets.filesystem.Remote attribute), 341	e <i>ESRespan<u>s</u>&amp;Back<u>et</u>system_image</i> ( <i>digi.xbee.profile.XBeeProfile attribute</i> ),
receive_options (digi.xbee.packets.raw.RX16IOPack	
attribute), 373	remote_port (digi.xbee.models.info.SocketInfo
<pre>attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet</pre>	remote_port (digi.xbee.models.info.SocketInfo attribute), 196
<pre>attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet     attribute), 364</pre>	remote_port (digi.xbee.models.info.SocketInfo attribute), 196 remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64I0Pack	remote_port (digi.xbee.models.info.SocketInfo attribute), 196 remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket et attribute), 420
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPack attribute), 367	remote_port (digi.xbee.models.info.SocketInfo attribute), 196 remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket et attribute), 420 RemoteATCmdOptions (class in
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64I0Pack	remote_port (digi.xbee.models.info.SocketInfo attribute), 196 remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket et attribute), 420 RemoteATCmdOptions (class in
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPack attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPack attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360 receive_options (digi.xbee.packets.wifi.IODataSamp	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360 receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360 receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433 receive_options (digi.xbee.packets.zigbee.OTAFirmv	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmw attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteRece attribute), 451	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360 receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433 receive_options (digi.xbee.packets.zigbee.OTAFirmwattribute), 461 receive_options (digi.xbee.packets.zigbee.RouteReceive_options (digi.xbee.packets.zigbee.RouteReceive_options (class in digi.xbee.models.options),	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmwattribute), 461  receive_options (digi.xbee.packets.zigbee.RouteReceive_options), 451  ReceiveOptions (class in digi.xbee.models.options), 208	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360 receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433 receive_options (digi.xbee.packets.zigbee.OTAFirmwattribute), 461 receive_options (digi.xbee.packets.zigbee.RouteReceive_options (digi.xbee.packets.zigbee.RouteReceive_options (class in digi.xbee.models.options),	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373 receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364 receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367 receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360 receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433 receive_options (digi.xbee.packets.zigbee.OTAFirmv attribute), 461 receive_options (digi.xbee.packets.zigbee.RouteReceive_options (digi.xbee.packets.zigbee.RouteReceive_options (class in digi.xbee.models.options), 208 ReceivePacket (class in digi.xbee.packets.common), 269	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmv attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteRece attribute), 451  ReceiveOptions (class in digi.xbee.models.options), 208  ReceivePacket (class in digi.xbee.packets.common),	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmv attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteRece attribute), 451  ReceiveOptions (class in digi.xbee.models.options), 208  ReceivePacket (class in digi.xbee.packets.common), 269  recover_device() (in module digi.xbee.recovery),	remote_port (digi.xbee.models.info.SocketInfo     attribute), 196  remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket et attribute), 420  RemoteATCmdOptions (class in     digi.xbee.models.options), 211  RemoteATCommandPacket (class in     leRxIndicatligWifbPaqkeckets.common), 273  RemoteATCommandResponsePacket (class in     vareUpdate&igixxlPackatkets.common), 277  RemoteATCommandResponseWifiPacket (class     ordIndicatoirPakiketxbee.packets.wifi), 439  RemoteATCommandWifiPacket (class in         digi.xbee.packets.wifi), 435  RemoteDigiMeshDevice (class in         digi.xbee.devices), 833  RemoteDigiPointDevice (class in         digi.xbee.devices), 852
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmv attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteRece attribute), 451  ReceiveOptions (class in digi.xbee.models.options), 208  ReceivePacket (class in digi.xbee.packets.common), 269  recover_device() (in module digi.xbee.recovery), 1028	remote_port (digi.xbee.models.info.SocketInfo     attribute), 196  remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket et attribute), 420  RemoteATCmdOptions (class in     digi.xbee.models.options), 211  RemoteATCommandPacket (class in     leexIndicattigWiftPaqkeckets.common), 273  RemoteATCommandResponsePacket (class in     vareUpdatedigiuxiPackatkets.common), 277  RemoteATCommandResponseWifiPacket (class     ordIndicatoitPakigitxbee.packets.wifi), 439  RemoteATCommandWifiPacket (class in         digi.xbee.packets.wifi), 435  RemoteDigiMeshDevice (class in         digi.xbee.devices), 833  RemoteDigiPointDevice (class in         digi.xbee.devices), 852  RemoteFSRequestPacket (class in
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmw attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteRece attribute), 451  ReceiveOptions (class in digi.xbee.models.options), 208  ReceivePacket (class in digi.xbee.packets.common), 269  recover_device() (in module digi.xbee.recovery), 1028  RecoveryException, 958	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmv attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteRece attribute), 451  ReceiveOptions (class in digi.xbee.models.options), 208  ReceivePacket (class in digi.xbee.packets.common), 269  recover_device() (in module digi.xbee.recovery), 1028  RecoveryException, 958  recv() (digi.xbee.xsocket.socket method), 1036  region_lock (digi.xbee.xsocket.socket method), 1036  region_lock (digi.xbee.profile.XBeeProfile attribute),	remote_port (digi.xbee.models.info.SocketInfo
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmv attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteReco attribute), 451  ReceiveOptions (class in digi.xbee.models.options), 208  ReceivePacket (class in digi.xbee.packets.common), 269  recover_device() (in module digi.xbee.recovery), 1028  RecoveryException, 958  recv() (digi.xbee.xsocket.socket method), 1035  recvfrom() (digi.xbee.xsocket.socket method), 1036  region_lock (digi.xbee.profile.XBeeProfile attribute), 993	remote_port (digi.xbee.models.info.SocketInfo     attribute), 196  remote_port (digi.xbee.packets.socket.SocketNewIPv4ClientPacket et attribute), 420  RemoteATCmdOptions (class in     digi.xbee.models.options), 211  RemoteATCommandPacket (class in  leRxIndicatdigWifbPaqkeckets.common), 273  RemoteATCommandResponsePacket (class in  vareUpdateGigintPackets.common), 277  RemoteATCommandResponseWifiPacket (class  ordIndicatoirPaligerxbee.packets.wifi), 439  RemoteATCommandWifiPacket (class in     digi.xbee.packets.wifi), 435  RemoteDigiMeshDevice (class in     digi.xbee.devices), 833  RemoteDigiPointDevice (class in     digi.xbee.devices), 852  RemoteFSRequestPacket (class in     digi.xbee.packets.filesystem), 335  RemoteFSResponsePacket (class in     digi.xbee.packets.filesystem), 338  RemoteRaw802Device (class in digi.xbee.devices),  814
attribute), 373  receive_options (digi.xbee.packets.raw.RX16Packet attribute), 364  receive_options (digi.xbee.packets.raw.RX64IOPacket attribute), 367  receive_options (digi.xbee.packets.raw.RX64Packet attribute), 360  receive_options (digi.xbee.packets.wifi.IODataSamp attribute), 433  receive_options (digi.xbee.packets.zigbee.OTAFirmv attribute), 461  receive_options (digi.xbee.packets.zigbee.RouteRece attribute), 451  ReceiveOptions (class in digi.xbee.models.options), 208  ReceivePacket (class in digi.xbee.packets.common), 269  recover_device() (in module digi.xbee.recovery), 1028  RecoveryException, 958  recv() (digi.xbee.xsocket.socket method), 1036  region_lock (digi.xbee.xsocket.socket method), 1036  region_lock (digi.xbee.profile.XBeeProfile attribute),	remote_port (digi.xbee.models.info.SocketInfo

871 remove\_device() (digi.xbee.devices.DigiPointNetwork (digi.xbee.filesystem.FileSystemManager remove() method), 949 *method*), 961 remove\_device() (digi.xbee.devices.Raw802Network (digi.xbee.reader.BluetoothDataReceived method), 925 remove() method), 1007 remove device() (digi.xbee.devices.XBeeNetwork (digi.xbee.reader.DataReceived method), *method*), 903 remove() 998 remove device() (digi.xbee.devices.ZigBeeNetwork remove() (digi.xbee.reader.DeviceDiscovered method), method), 913 remove\_element() (digi.xbee.filesystem.LocalXBeeFileSystemManage 1002 remove() (digi.xbee.reader.DiscoveryProcessFinished method), 973 method), 1002 RenameCmdRequest (class in(digi.xbee.reader.EndDiscoveryScan digi.xbee.models.filesystem), 178 remove() method), 1015 RenameCmdResponse (class in (digi.xbee.reader.ExplicitDataReceived remove() digi.xbee.models.filesystem), 180  ${\tt REPEATER\_MODE}\ (digi.xbee.models.options. Receive Options$ method), 1003 remove() (digi.xbee.reader.FileSystemFrameReceived attribute), 209 REPEATER\_MODE (digi.xbee.models.options.TransmitOptions method), 1016 (digi.xbee.reader.InitDiscoveryScan attribute), 211 remove()  $\verb|request_data| (\textit{digi.xbee.packets.devicecloud.DeviceRequestPacket}|$ method), 1014 remove() (digi.xbee.reader.IOSampleReceived attribute), 307 method), 1000 request\_data(digi.xbee.packets.devicecloud.DeviceResponsePacket remove() (digi.xbee.reader.IPDataReceived method), attribute), 310 1004 request\_id(digi.xbee.packets.devicecloud.DeviceRequestPacket remove() (digi.xbee.reader.MicroPythonDataReceived attribute), 307 method), 1008 request\_id(digi.xbee.packets.devicecloud.DeviceResponsePacket remove() (digi.xbee.reader.ModemStatusReceived attribute), 310 method), 999 (digi.xbee.devices.AbstractXBeeDevice reset() remove() (digi.xbee.reader.NetworkModified method), method), 476 1001 (digi.xbee.devices.CellularDevice method), reset() remove() (digi.xbee.reader.NetworkUpdateProgress 699 *method*), 1016 reset() (digi.xbee.devices.DigiMeshDevice method), (digi.xbee.reader.PacketReceived method), 572 remove() 996 reset() (digi.xbee.devices.DigiPointDevice method), (digi.xbee.reader.PacketReceivedFrom remove() method), 997 reset () (digi.xbee.devices.IPDevice method), 672 (digi.xbee.reader.RelayDataReceived (digi.xbee.devices.LPWANDevice method), remove() reset() method), 1006 726 remove() (digi.xbee.reader.RouteInformationReceived reset () (digi.xbee.devices.NBIoTDevice method), 752 method), 1012 (digi.xbee.devices.Raw802Device method), reset() remove() (digi.xbee.reader.RouteReceived method), 539 1013 (digi.xbee.devices.RemoteDigiMeshDevice reset() remove() (digi.xbee.reader.RouteRecordIndicatorReceived method), 845 (digi.xbee.devices.RemoteDigiPointDevice method), 1011 reset() (digi.xbee.reader.SMSReceived method), method), 863 remove() (digi.xbee.devices.RemoteRaw802Device 1005 reset() (digi.xbee.reader.SocketDataReceived method), 825 remove() method), 1009 reset() (digi.xbee.devices.RemoteXBeeDevice (digi.xbee.reader.SocketDataReceivedFrom remove() method), 796 method), 1010 (digi.xbee.devices.RemoteZigBeeDevice reset() (digi.xbee.reader.SocketStateReceived method), 884 remove() reset() (digi.xbee.devices.WiFiDevice method), 784 method), 1008 remove() (digi.xbee.reader.XBeeEvent method), 995 reset () (digi.xbee.devices.XBeeDevice method), 497 remove\_device() (digi.xbee.devices.DigiMeshNetworkreset() (digi.xbee.devices.ZigBeeDevice method), 642

1110 Index

*method*), 937

reset settings (digi.xbee.profile.XBeeProfile at-

tribute), 991	reverse() (digi.xbee.reader.SocketDataReceivedFrom
responder_addr (digi.xbee.packets.digimesh.RouteInfo	ormationPa <b>vket</b> hod), 1010
attribute), 328	reverse() (digi.xbee.reader.SocketStateReceived
response (digi.xbee.models.atcomm.ATCommandRespon	
attribute), 135	reverse() (digi.xbee.reader.XBeeEvent method), 995
restore_after_update()	rf_data(digi.xbee.packets.common.ExplicitAddressingPacket
(digi.xbee.firmware.UpdateConfigurer method), 976	attribute), 300 rf_data(digi.xbee.packets.common.ExplicitRXIndicatorPacket
restore_total (digi.xbee.firmware.UpdateConfigurer	attribute), 305
attribute), 976	rf_data(digi.xbee.packets.common.IODataSampleRxIndicatorPacket
reverse() (digi.xbee.reader.BluetoothDataReceived	attribute), 294
method), 1007	rf_data (digi.xbee.packets.common.ReceivePacket at-
reverse() (digi.xbee.reader.DataReceived method),	tribute), 271
998	rf_data(digi.xbee.packets.common.TransmitPacket at-
reverse() (digi.xbee.reader.DeviceDiscovered	tribute), 283
method), 1002	rf_data (digi.xbee.packets.raw.RX16IOPacket at-
reverse() (digi.xbee.reader.DiscoveryProcessFinished	tribute), 373
method), 1003	rf_data (digi.xbee.packets.raw.RX16Packet attribute), 364
reverse() (digi.xbee.reader.EndDiscoveryScan method), 1015	rf_data (digi.xbee.packets.raw.RX64IOPacket at-
reverse() (digi.xbee.reader.ExplicitDataReceived	tribute), 368
method), 1003	rf_data (digi.xbee.packets.raw.RX64Packet attribute),
reverse() (digi.xbee.reader.FileSystemFrameReceived	361
method), 1016	rf_data (digi.xbee.packets.raw.TX16Packet attribute),
reverse() (digi.xbee.reader.InitDiscoveryScan	354
method), 1014	rf_data (digi.xbee.packets.raw.TX64Packet attribute),
reverse() (digi.xbee.reader.IOSampleReceived	351
method), 1000	rf_data(digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiPacket
reverse() (digi.xbee.reader.IPDataReceived method),	attribute), 433
1004	Role (class in digi.xbee.models.protocol), 217
reverse() (digi.xbee.reader.MicroPythonDataReceived method), 1008	234
	Route (class in digi.xbee.models.zdo), 237
method), 999	route_cmd_options
reverse() (digi.xbee.reader.NetworkModified	digi.xbee.packets.zigbee.CreateSourceRoutePacket
method), 1001	attribute), 455
$\verb reverse()  (\textit{digi.xbee.reader.NetworkUpdateProgress} $	
method), 1017	digi.xbee.packets.digimesh), 324
reverse() (digi.xbee.reader.PacketReceived method),	RouteInformationReceived (class in
996	digi.xbee.reader), 1011
reverse() (digi.xbee.reader.PacketReceivedFrom	RouteReceived (class in digi.xbee.reader), 1012
method), 997 reverse() (digi.xbee.reader.RelayDataReceived	RouteRecordIndicatorPacket (class in digi.xbee.packets.zigbee), 449
method), 1006	RouteRecordIndicatorReceived (class in
reverse() (digi.xbee.reader.RouteInformationReceived	digi.xbee.reader), 1010
method), 1012	RouteStatus (class in digi.xbee.models.zdo), 237
reverse() (digi.xbee.reader.RouteReceived method),	RouteTableReader (class in digi.xbee.models.zdo),
1013	235
$\verb reverse  ()   \textit{(digi.xbee.reader.RouteRecordIndicatorReceived)}     \textit{(digi.xbee.reader.RouteRecordIndicatorReceived)}                                      $	
method), 1011	372
reverse() (digi.xbee.reader.SMSReceived method),	rssi (digi.xbee.packets.raw.RX16Packet attribute), 364
1005	rssi (digi.xbee.packets.raw.RX64IOPacket attribute),
reverse() (digi.xbee.reader.SocketDataReceived method), 1009	367 rssi (digi.xbee.packets.raw.RX64Packet attribute), 360
пинои), 1007	1551 (aiginoccipacionawinaoti achei annionie), 500

```
rssi (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifiBackat_counter (digi.xbee.devices.RemoteZigBeeDevice
         attribute), 433
                                                              attribute), 884
run () (digi.xbee.reader.PacketListener method), 1017
                                                     scan counter
                                                                        (digi.xbee.devices.WiFiDevice
running (digi.xbee.filesystem.FileProcess attribute),
                                                              tribute), 784
        960
                                                     scan counter
                                                                        (digi.xbee.devices.XBeeDevice
running (digi.xbee.models.zdo.NeighborFinder
                                                              tribute), 512
        tribute), 241
                                                     scan counter (digi.xbee.devices.XBeeNetwork at-
running (digi.xbee.models.zdo.NeighborTableReader
                                                              tribute), 892
        attribute), 239
                                                     scan_counter (digi.xbee.devices.ZigBeeDevice at-
running (digi.xbee.models.zdo.NodeDescriptorReader
                                                              tribute), 643
        attribute), 233
                                                     scan_counter (digi.xbee.devices.ZigBeeNetwork at-
                                                              tribute), 913
running (digi.xbee.models.zdo.RouteTableReader at-
        tribute), 237
                                                     scan_counter_a2b
                                                                              (digi.xbee.devices.Connection
RX16IOPacket (class in digi.xbee.packets.raw), 369
                                                              attribute), 955
RX16Packet (class in digi.xbee.packets.raw), 362
                                                     scan_counter_b2a
                                                                              (digi.xbee.devices.Connection
RX64IOPacket (class in digi.xbee.packets.raw), 366
                                                              attribute), 955
RX64Packet (class in digi.xbee.packets.raw), 359
                                                     SCAN_TIL_CANCEL (digi.xbee.devices.XBeeNetwork
RXIPv4Packet (class in digi.xbee.packets.network),
                                                              attribute), 892
                                                     SECURE (digi.xbee.models.options.FileOpenRequestOption
RXSMSPacket (class in digi.xbee.packets.cellular), 253
                                                              attribute), 215
                                                     SECURE_SESSION_ENC
S
                                                              (digi.xbee.models.options.ReceiveOptions
                                                              attribute), 209
scan_access_points()
                                                     SECURE SESSION ENC
        (digi.xbee.devices.WiFiDevice method), 763
                                                              (digi.xbee.models.options.RemoteATCmdOptions
scan_counter(digi.xbee.devices.AbstractXBeeDevice
                                                              attribute), 212
        attribute), 491
                                                     SECURE_SESSION_ENC
scan_counter (digi.xbee.devices.CellularDevice at-
                                                              (digi.xbee.models.options.TransmitOptions
        tribute), 699
                                                              attribute), 211
scan_counter (digi.xbee.devices.DigiMeshDevice at-
                                                     send() (digi.xbee.sender.SyncRequestSender method),
        tribute), 572
                                                              1029
scan_counter (digi.xbee.devices.DigiMeshNetwork
                                                     send() (digi.xbee.xsocket.socket method), 1036
        attribute), 937
                                                     send_bluetooth_data()
scan_counter (digi.xbee.devices.DigiPointDevice at-
                                                              (digi.xbee.devices.CellularDevice
                                                                                                 method),
        tribute), 604
                                                              699
scan_counter (digi.xbee.devices.DigiPointNetwork
                                                     send_bluetooth_data()
        attribute), 949
                                                              (digi.xbee.devices.DigiMeshDevice
                                                                                                 method),
scan_counter (digi.xbee.devices.IPDevice attribute),
                                                              572
        672
                                                     send_bluetooth_data()
scan_counter (digi.xbee.devices.LPWANDevice at-
                                                              (digi.xbee.devices.DigiPointDevice
                                                                                                 method),
        tribute), 726
                      (digi.xbee.devices.NBIoTDevice
scan counter
                                                     send_bluetooth_data()
        attribute), 752
                                                              (digi.xbee.devices.IPDevice method), 672
scan_counter (digi.xbee.devices.Raw802Device at-
                                                     send_bluetooth_data()
        tribute), 539
                                                              (digi.xbee.devices.LPWANDevice
                                                                                                 method),
scan_counter (digi.xbee.devices.Raw802Network at-
                                                              726
        tribute), 925
scan_counter(digi.xbee.devices.RemoteDigiMeshDeviceend_bluetooth_data()
                                                              (digi.xbee.devices.NBIoTDevice
                                                                                                 method),
        attribute), 845
                                                              752
scan_counter(digi.xbee.devices.RemoteDigiPointDevice
                                                     send_bluetooth_data()
        attribute), 863
                                                              (digi.xbee.devices.Raw802Device
                                                                                                 method),
scan_counter(digi.xbee.devices.RemoteRaw802Device
        attribute), 825
                                                     send_bluetooth_data()
scan_counter (digi.xbee.devices.RemoteXBeeDevice
                                                              (digi.xbee.devices.WiFiDevice method), 784
        attribute), 807
```

```
753
send bluetooth data()
        (digi.xbee.devices.XBeeDevice
                                          method).
                                                    send_data_async()
        495
                                                            (digi.xbee.devices.Raw802Device
                                                                                              method),
send_bluetooth_data()
        (digi.xbee.devices.ZigBeeDevice
                                          method),
                                                    send_data_async()
                                                                           (digi.xbee.devices.WiFiDevice
                                                            method), 785
                    (digi.xbee.devices.CellularDevice
                                                    send data async() (digi.xbee.devices.XBeeDevice
send data()
        method), 699
                                                            method), 494
send_data()
                   (digi.xbee.devices.DigiMeshDevice
                                                    send_data_async()
        method), 572
                                                            (digi.xbee.devices.ZigBeeDevice
                                                                                              method),
                   (digi.xbee.devices.DigiPointDevice
send_data()
                                                            643
                                                    send_data_async_16()
        method), 605
               (digi.xbee.devices.IPDevice method),
                                                            (digi.xbee.devices.Raw802Device
                                                                                              method),
send_data()
                                                            522
        658
                    (digi.xbee.devices.LPWANDevice
send_data()
                                                    send_data_async_64()
        method), 727
                                                            (digi.xbee.devices.DigiMeshDevice
                                                                                              method),
                                                            551
send_data()
                      (digi.xbee.devices.NBIoTDevice
        method), 753
                                                    send_data_async_64()
send_data()
                    (digi.xbee.devices.Raw802Device
                                                            (digi.xbee.devices.Raw802Device
                                                                                              method),
        method), 539
                                                            521
send_data() (digi.xbee.devices.WiFiDevice method),
                                                    send_data_async_64_16()
                                                            (digi.xbee.devices.DigiPointDevice
                                                                                              method),
                                                            584
send_data() (digi.xbee.devices.XBeeDevice method),
        493
                                                    send data async 64 16()
send data()
                     (digi.xbee.devices.ZigBeeDevice
                                                            (digi.xbee.devices.ZigBeeDevice
                                                                                              method),
        method), 643
                                                            617
send_data_16()
                    (digi.xbee.devices.Raw802Device
                                                    send_data_broadcast()
                                                            (digi.xbee.devices.CellularDevice
        method), 521
                                                                                              method),
send_data_64() (digi.xbee.devices.DigiMeshDevice
        method), 551
                                                    send_data_broadcast()
send_data_64()
                    (digi.xbee.devices.Raw802Device
                                                            (digi.xbee.devices.DigiMeshDevice
                                                                                              method),
        method), 520
                                                            573
send_data_64_16()
                                                    send_data_broadcast()
        (digi.xbee.devices.DigiPointDevice
                                                            (digi.xbee.devices.DigiPointDevice
                                          method),
                                                                                              method),
        583
                                                            606
                                                    send data broadcast()
send_data_64_16()
        (digi.xbee.devices.ZigBeeDevice
                                          method),
                                                            (digi.xbee.devices.IPDevice method), 658
        617
                                                    send_data_broadcast()
                                                            (digi.xbee.devices.LPWANDevice
                                                                                              method),
send_data_async()
        (digi.xbee.devices.CellularDevice
                                                            727
                                          method),
        699
                                                    send data broadcast()
send_data_async()
                                                            (digi.xbee.devices.NBIoTDevice
                                                                                              method),
        (digi.xbee.devices.DigiMeshDevice
                                          method),
        573
                                                    send_data_broadcast()
send_data_async()
                                                            (digi.xbee.devices.Raw802Device
                                                                                              method),
                                                            540
        (digi.xbee.devices.DigiPointDevice
                                          method),
        605
                                                    send_data_broadcast()
                                                            (digi.xbee.devices.WiFiDevice method), 785
send_data_async()
                         (digi.xbee.devices.IPDevice
        method), 658
                                                    send_data_broadcast()
send_data_async()
                                                            (digi.xbee.devices.XBeeDevice
                                                                                              method),
        (digi.xbee.devices.LPWANDevice
                                                            494
                                          method),
        72.7
                                                    send data broadcast()
send_data_async()
                                                            (digi.xbee.devices.ZigBeeDevice
                                                                                              method),
        (digi.xbee.devices.NBIoTDevice
                                          method),
                                                            644
```

```
send_expl_data() (digi.xbee.devices.DigiMeshDevice send_ip_data_broadcast()
        method), 553
                                                            (digi.xbee.devices.IPDevice method), 656
send_expl_data() (digi.xbee.devices.DigiPointDevice send_ip_data_broadcast()
        method), 586
                                                            (digi.xbee.devices.LPWANDevice
                                                                                            method),
send_expl_data() (digi.xbee.devices.ZigBeeDevice
        method), 619
                                                   send ip data broadcast()
send expl data async()
                                                            (digi.xbee.devices.NBIoTDevice
                                                                                            method),
        (digi.xbee.devices.DigiMeshDevice
                                         method),
        554
                                                   send_ip_data_broadcast()
                                                            (digi.xbee.devices.WiFiDevice method), 786
send_expl_data_async()
        (digi.xbee.devices.DigiPointDevice
                                         method),
                                                   send_micropython_data()
                                                            (digi.xbee.devices.CellularDevice
                                                                                            method),
send_expl_data_async()
        (digi.xbee.devices.ZigBeeDevice
                                                   send_micropython_data()
                                         method),
                                                            (digi.xbee.devices.DigiMeshDevice
        621
                                                                                            method),
send_expl_data_broadcast()
                                                            574
        (digi.xbee.devices.DigiMeshDevice
                                         method),
                                                   send_micropython_data()
        554
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                            method),
send_expl_data_broadcast()
        (digi.xbee.devices.DigiPointDevice
                                         method),
                                                   send micropython data()
                                                            (digi.xbee.devices.IPDevice method), 673
send_expl_data_broadcast()
                                                   send_micropython_data()
        (digi.xbee.devices.ZigBeeDevice
                                                           (digi.xbee.devices.LPWANDevice
                                         method),
                                                                                            method),
                                                            727
        620
send_file_xmodem()
                                (in
                                           module
                                                   send_micropython_data()
        digi.xbee.util.xmodem), 468
                                                           (digi.xbee.devices.NBIoTDevice
                                                                                            method),
send_file_ymodem()
                               (in
                                           module
                                                           754
        digi.xbee.util.xmodem), 469
                                                   send_micropython_data()
                    (digi.xbee.devices.CellularDevice
                                                           (digi.xbee.devices.Raw802Device
send_ip_data()
                                                                                            method),
        method), 700
                                                            541
send_ip_data()
                         (digi.xbee.devices.IPDevice
                                                   send_micropython_data()
        method), 655
                                                            (digi.xbee.devices.WiFiDevice method), 786
send_ip_data()
                    (digi.xbee.devices.LPWANDevice
                                                   send_micropython_data()
        method), 709
                                                           (digi.xbee.devices.XBeeDevice
                                                                                            method),
send_ip_data()
                     (digi.xbee.devices.NBIoTDevice
        method), 753
                                                   send_micropython_data()
send_ip_data()
                       (digi.xbee.devices.WiFiDevice
                                                           (digi.xbee.devices.ZigBeeDevice
                                                                                            method),
        method), 785
                                                            644
send_ip_data_async()
                                                   send_multicast_data()
        (digi.xbee.devices.CellularDevice
                                                           (digi.xbee.devices.ZigBeeDevice
                                         method),
                                                                                            method),
        700
send_ip_data_async()
                                                   send_multicast_data_async()
        (digi.xbee.devices.IPDevice method), 656
                                                           (digi.xbee.devices.ZigBeeDevice
                                                                                            method),
                                                            622
send_ip_data_async()
        (digi.xbee.devices.LPWANDevice
                                         method),
                                                   send_packet()
                                                                       (digi.xbee.devices.CellularDevice
        709
                                                            method), 701
send_ip_data_async()
                                                   send_packet()
                                                                      (digi.xbee.devices.DigiMeshDevice
        (digi.xbee.devices.NBIoTDevice
                                         method),
                                                           method), 574
        753
                                                   send_packet()
                                                                      (digi.xbee.devices.DigiPointDevice
send_ip_data_async()
                                                           method), 606
        (digi.xbee.devices.WiFiDevice method), 785
                                                   send_packet() (digi.xbee.devices.IPDevice method),
send ip data broadcast()
                                                           673
        (digi.xbee.devices.CellularDevice
                                         method),
                                                   send_packet()
                                                                       (digi.xbee.devices.LPWANDevice
        700
                                                           method), 727
```

1 1 1 () (I'' I I ' MINITED '	(07
send_packet() (digi.xbee.devices.NBIoTDevice	607
method), 754	send_user_data_relay()
send_packet() (digi.xbee.devices.Raw802Device	(digi.xbee.devices.IPDevice method), 674
method), 541	send_user_data_relay()
send_packet() (digi.xbee.devices.WiFiDevice	(digi.xbee.devices.LPWANDevice method),
method), 786	728
send_packet() (digi.xbee.devices.XBeeDevice	send_user_data_relay()
method), 501	(digi.xbee.devices.NBIoTDevice method),
send_packet() (digi.xbee.devices.ZigBeeDevice	755
method), 645	send_user_data_relay()
send_packet() (digi.xbee.sender.PacketSender	(digi.xbee.devices.Raw802Device method),
method), 1028	542
<pre>send_packet_sync_and_get_response()</pre>	send_user_data_relay()
(digi.xbee.devices.CellularDevice method), 702	(digi.xbee.devices.WiFiDevice method), 787
send_packet_sync_and_get_response()	send_user_data_relay()
(digi.xbee.devices.DigiMeshDevice method),	(digi.xbee.devices.XBeeDevice method),
575	495
<pre>send_packet_sync_and_get_response()</pre>	send_user_data_relay()
(digi.xbee.devices.DigiPointDevice method),	(digi.xbee.devices.ZigBeeDevice method),
607	646
<pre>send_packet_sync_and_get_response()</pre>	sendall() (digi.xbee.xsocket.socket method), 1036
(digi.xbee.devices.IPDevice method), 673	SendDataRequestOptions (class in
<pre>send_packet_sync_and_get_response()</pre>	digi.xbee.models.options), 212
(digi.xbee.devices.LPWANDevice method), 728	SendDataRequestPacket (class in
<pre>send_packet_sync_and_get_response()</pre>	digi.xbee.packets.devicecloud), 318
(digi.xbee.devices.NBIoTDevice method), 755	SendDataResponsePacket (class in
<pre>send_packet_sync_and_get_response()</pre>	digi.xbee.packets.devicecloud), 321
(digi.xbee.devices.Raw802Device method), 542	sendto() (digi.xbee.xsocket.socket method), 1037
send_packet_sync_and_get_response()	SENT_FROM_END_DEVICE
(digi.xbee.devices.WiFiDevice method), 787	(digi.xbee.models.options.ReceiveOptions
send_packet_sync_and_get_response()	attribute), 209
(digi.xbee.devices.XBeeDevice method), 501	<pre>serial_port (digi.xbee.devices.CellularDevice at-</pre>
send_packet_sync_and_get_response()	tribute), 702
(digi.xbee.devices.ZigBeeDevice method), 645	<pre>serial_port (digi.xbee.devices.DigiMeshDevice at-</pre>
send_sms() (digi.xbee.devices.CellularDevice	tribute), 575
method), 682	serial_port (digi.xbee.devices.DigiPointDevice at-
send_sms() (digi.xbee.devices.LPWANDevice	tribute), 608
method), 709	serial_port (digi.xbee.devices.IPDevice attribute),
send_sms() (digi.xbee.devices.NBIoTDevice method),	674
755	serial_port (digi.xbee.devices.LPWANDevice
send_sms_async() (digi.xbee.devices.CellularDevice	attribute), 729
method), 683	serial_port (digi.xbee.devices.NBIoTDevice at-
send_sms_async() (digi.xbee.devices.LPWANDevice	tribute), 756
method), 709	serial_port (digi.xbee.devices.Raw802Device
send_sms_async() (digi.xbee.devices.NBIoTDevice	attribute), 542
method), 755	serial_port (digi.xbee.devices.WiFiDevice at-
send_user_data_relay()	tribute), 788
(digi.xbee.devices.CellularDevice method),	serial_port (digi.xbee.devices.XBeeDevice at-
702	tribute), 493
send_user_data_relay()	serial_port (digi.xbee.devices.ZigBeeDevice at-
(digi.xbee.devices.DigiMeshDevice method),	tribute), 646
575	set_16bit_addr() (digi.xbee.devices.AbstractXBeeDevice
send_user_data_relay()	method), 478
(digi.xbee.devices.DigiPointDevice method),	set_16bit_addr() (digi.xbee.devices.CellularDevice
(aigi.nocc.acvices.Digit oniDevice memou),	SCCLOSCada_ () (algwcc.acvaces.Cenambevace

```
method), 703
                                                           (digi.xbee.devices.Raw802Device
                                                                                            method),
set_16bit_addr() (digi.xbee.devices.DigiMeshDevice
                                                           543
        method), 576
                                                   set api output mode()
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
set_16bit_addr() (digi.xbee.devices.DigiPointDevice
        method), 608
                                                           method), 845
                         (digi.xbee.devices.IPDevice
                                                  set api output mode()
set 16bit addr()
                                                           (digi.xbee.devices.RemoteDigiPointDevice
        method), 675
set_16bit_addr() (digi.xbee.devices.LPWANDevice
                                                           method), 864
        method), 729
                                                   set_api_output_mode()
                                                           (digi.xbee.devices.RemoteRaw802Device
set_16bit_addr() (digi.xbee.devices.NBIoTDevice
        method), 756
                                                           method), 826
set_16bit_addr() (digi.xbee.devices.Raw802Device set_api_output_mode()
                                                           (digi.xbee.devices.RemoteXBeeDevice
        method), 543
set_16bit_addr() (digi.xbee.devices.RemoteDigiMeshDevice
                                                           method), 807
        method), 845
                                                   set_api_output_mode()
set_16bit_addr() (digi.xbee.devices.RemoteDigiPointDevice
                                                           (digi.xbee.devices.RemoteZigBeeDevice
        method), 864
                                                           method), 885
set_16bit_addr() (digi.xbee.devices.RemoteRaw802Device_api_output_mode()
                                                           (digi.xbee.devices.WiFiDevice method), 788
        method), 825
set 16bit addr() (digi.xbee.devices.RemoteXBeeDeviset api output mode()
        method), 807
                                                           (digi.xbee.devices.XBeeDevice
                                                                                            method),
set_16bit_addr() (digi.xbee.devices.RemoteZigBeeDevice
        method), 884
                                                   set_api_output_mode()
                       (digi.xbee.devices.WiFiDevice
                                                           (digi.xbee.devices.ZigBeeDevice
set 16bit addr()
                                                                                            method).
                                                           647
        method), 788
set_16bit_addr()
                                                  set_api_output_mode_value()
                      (digi.xbee.devices.XBeeDevice
        method), 513
                                                           (digi.xbee.devices.AbstractXBeeDevice
set_16bit_addr() (digi.xbee.devices.ZigBeeDevice
                                                           method), 488
        method), 646
                                                   set_api_output_mode_value()
set_64bit_addr() (digi.xbee.devices.RemoteRaw802Device
                                                           (digi.xbee.devices.CellularDevice
                                                                                            method),
        method), 814
                                                           703
set_access_point_timeout()
                                                   set_api_output_mode_value()
        (digi.xbee.devices.WiFiDevice method), 766
                                                           (digi.xbee.devices.DigiMeshDevice
                                                                                            method),
set_api_output_mode()
        (digi.xbee.devices.AbstractXBeeDevice
                                                   set api output mode value()
        method), 487
                                                           (digi.xbee.devices.DigiPointDevice
                                                                                            method),
set_api_output_mode()
                                                           609
        (digi.xbee.devices.CellularDevice
                                         method),
                                                   set_api_output_mode_value()
        703
                                                           (digi.xbee.devices.IPDevice method), 675
                                                   set_api_output_mode_value()
set_api_output_mode()
        (digi.xbee.devices.DigiMeshDevice
                                         method),
                                                           (digi.xbee.devices.LPWANDevice
                                                                                            method),
        576
                                                           730
set_api_output_mode()
                                                   set_api_output_mode_value()
        (digi.xbee.devices.DigiPointDevice
                                                           (digi.xbee.devices.NBIoTDevice
                                         method),
                                                                                            method),
                                                   set_api_output_mode_value()
set_api_output_mode()
                                                           (digi.xbee.devices.Raw802Device
        (digi.xbee.devices.IPDevice method), 675
                                                                                            method),
set_api_output_mode()
                                                           543
        (digi.xbee.devices.LPWANDevice
                                         method),
                                                   set_api_output_mode_value()
                                                           (digi.xbee.devices.RemoteDigiMeshDevice
                                                           method), 845
set_api_output_mode()
        (digi.xbee.devices.NBIoTDevice
                                         method),
                                                  set_api_output_mode_value()
        756
                                                           (digi.xbee.devices.RemoteDigiPointDevice
set_api_output_mode()
                                                           method), 864
```

set_	_api_output_mode_value()		704
	(digi.xbee.devices.RemoteRaw802Dev	vice	set_dest_address()
	method), 826		(digi.xbee.devices.DigiMeshDevice method)
set_	_api_output_mode_value()		577
	(digi.xbee.devices.RemoteXBeeDevice	2	<pre>set_dest_address()</pre>
	method), 808		(digi.xbee.devices.DigiPointDevice method)
set_	_api_output_mode_value()		609
	(digi.xbee.devices.RemoteZigBeeDevi	ce	<pre>set_dest_address() (digi.xbee.devices.IPDevices.</pre>
	method), 885		method), 657
set.	_api_output_mode_value()		set_dest_address()
-	(digi.xbee.devices.WiFiDevice method	n 789	(digi.xbee.devices.LPWANDevice method)
a o +		1), 70)	730
set_	_api_output_mode_value()	d D	
	(digi.xbee.devices.XBeeDevice	method),	set_dest_address()
	513		(digi.xbee.devices.NBIoTDevice method)
set_	_api_output_mode_value()		757
	(digi.xbee.devices.ZigBeeDevice	method),	set_dest_address()
	647		(digi.xbee.devices.Raw802Device method)
set.	_baudrate() (digi.xbee.serial.XBee.	SerialPort	544
	method), 1032		set_dest_address()
cot	_deep_discovery_options()		(digi.xbee.devices.RemoteDigiMeshDevice
366_		mathad)	
	(digi.xbee.devices.DigiMeshNetwork	meinoa),	method), 846
	937		set_dest_address()
set_	_deep_discovery_options()		(digi.xbee.devices.RemoteDigiPointDevice
	(digi.xbee.devices.DigiPointNetwork	method),	method), 865
	949		set_dest_address()
set	_deep_discovery_options()		(digi.xbee.devices.RemoteRaw802Device
_	(digi.xbee.devices.Raw802Network	method),	method), 827
	925	,	set_dest_address()
ao+	_deep_discovery_options()		(digi.xbee.devices.RemoteXBeeDevice
set_		d D	
	(digi.xbee.devices.XBeeNetwork	method),	method), 808
	900		set_dest_address()
set_	_deep_discovery_options()		(digi.xbee.devices.RemoteZigBeeDevice
	(digi.xbee.devices.ZigBeeNetwork	method),	method), 885
	913		set_dest_address()
set	_deep_discovery_timeouts()		(digi.xbee.devices.WiFiDevice method), 789
	(digi.xbee.devices.DigiMeshNetwork	method).	set_dest_address()
	938	,,	(digi.xbee.devices.XBeeDevice method)
cat	_deep_discovery_timeouts()		514
366_		mathad)	
	(digi.xbee.devices.DigiPointNetwork	meinoa),	
	950		(digi.xbee.devices.ZigBeeDevice method)
set_	_deep_discovery_timeouts()		647
	(digi.xbee.devices.Raw802Network	method),	set_dest_ip_addr()
	926		(digi.xbee.devices.CellularDevice method)
set_	_deep_discovery_timeouts()		704
	(digi.xbee.devices.XBeeNetwork	method),	<pre>set_dest_ip_addr() (digi.xbee.devices.IPDevices.</pre>
	901		method), 654
sat	_deep_discovery_timeouts()		set_dest_ip_addr()
500_		mathad)	
	(digi.xbee.devices.ZigBeeNetwork	method),	(digi.xbee.devices.LPWANDevice method)
	914		730
set_	_dest_address()		set_dest_ip_addr()
	(digi.xbee.devices.AbstractXBeeDevic	e e	(digi.xbee.devices.NBIoTDevice method)
	method), 480		757
set_	_dest_address()		set_dest_ip_addr()
	(digi.xbee.devices.CellularDevice	method),	(digi.xbee.devices.WiFiDevice method), 789

set_dio_change_detection()  (disi_rhan_davings_AbstractYRanDavings_	<pre>set_dio_value()</pre>
(digi.xbee.devices.AbstractXBeeDevice method), 486	set_dio_value() (digi.xbee.devices.LPWANDevice
set_dio_change_detection()	method), 731
(digi.xbee.devices.CellularDevice method), 684	<pre>set_dio_value() (digi.xbee.devices.NBIoTDevice     method), 757</pre>
set_dio_change_detection()	set_dio_value() (digi.xbee.devices.Raw802Device
(digi.xbee.devices.DigiMeshDevice method), 577	<pre>method), 545 set_dio_value() (digi.xbee.devices.RemoteDigiMeshDevice</pre>
set_dio_change_detection()	method), 847
(digi.xbee.devices.DigiPointDevice method),	set_dio_value() (digi.xbee.devices.RemoteDigiPointDevice
610	method), 865
<pre>set_dio_change_detection()     (digi.xbee.devices.IPDevice method), 676</pre>	<pre>set_dio_value() (digi.xbee.devices.RemoteRaw802Device</pre>
<pre>set_dio_change_detection()</pre>	set_dio_value() (digi.xbee.devices.RemoteXBeeDevice
(digi.xbee.devices.LPWANDevice method),	method), 809
731	set_dio_value() (digi.xbee.devices.RemoteZigBeeDevice
<pre>set_dio_change_detection()      (digi.xbee.devices.NBIoTDevice method),</pre>	<pre>method), 886 set_dio_value() (digi.xbee.devices.WiFiDevice</pre>
757	method), 790
<pre>set_dio_change_detection()</pre>	set_dio_value() (digi.xbee.devices.XBeeDevice
(digi.xbee.devices.Raw802Device method),	method), 514
544	set_dio_value() (digi.xbee.devices.ZigBeeDevice
set_dio_change_detection()	method), 648
(digi.xbee.devices.RemoteDigiMeshDevice method), 846	<pre>set_discovery_options()     (digi.xbee.devices.DigiMeshNetwork method),</pre>
set_dio_change_detection()	938
(digi.xbee.devices.RemoteDigiPointDevice	set_discovery_options()
method), 865	(digi.xbee.devices.DigiPointNetwork method),
<pre>set_dio_change_detection()</pre>	950
(digi.xbee.devices.RemoteRaw802Device	set_discovery_options()
method), 827	(digi.xbee.devices.Raw802Network method),
set_dio_change_detection()	926
(digi.xbee.devices.RemoteXBeeDevice method), 809	<pre>set_discovery_options()     (digi.xbee.devices.XBeeNetwork method),</pre>
set_dio_change_detection()	899
(digi.xbee.devices.RemoteZigBeeDevice	set_discovery_options()
method), 886	(digi.xbee.devices.ZigBeeNetwork method),
<pre>set_dio_change_detection()</pre>	914
(digi.xbee.devices.WiFiDevice method), 790	set_discovery_timeout()
set_dio_change_detection()	(digi.xbee.devices.DigiMeshNetwork method),
(digi.xbee.devices.XBeeDevice method), 514	938 set_discovery_timeout()
set_dio_change_detection()	(digi.xbee.devices.DigiPointNetwork method),
(digi.xbee.devices.ZigBeeDevice method),	950
648	<pre>set_discovery_timeout()</pre>
<pre>set_dio_value() (digi.xbee.devices.AbstractXBeeDev method), 485</pre>	ice (digi.xbee.devices.Raw802Network method), 926
set_dio_value() (digi.xbee.devices.CellularDevice	set_discovery_timeout()
method), 704	(digi.xbee.devices.XBeeNetwork method),
<pre>set_dio_value() (digi.xbee.devices.DigiMeshDevice</pre>	900 set_discovery_timeout()
set_dio_value() (digi.xbee.devices.DigiPointDevice	(digi.xbee.devices.ZigBeeNetwork method),
method), 610	914

set_dn	<pre>s_address() (digi.xbee.devices.W method), 768</pre>	'iFiDevice	(digi.xbee.devices.CellularDevice method) 684
_	teway_address() (digi.xbee.devices.WiFiDevice method _configuration()	d), 768	<pre>set_io_sampling_rate()      (digi.xbee.devices.DigiMeshDevice method) 578</pre>
	(digi.xbee.devices.AbstractXBeeDevice method), 481	ce	<pre>set_io_sampling_rate()      (digi.xbee.devices.DigiPointDevice method)</pre>
set_io	_configuration() (digi.xbee.devices.CellularDevice 705	method),	611 set_io_sampling_rate() (digi.xbee.devices.IPDevice method), 677
set_io	_configuration() (digi.xbee.devices.DigiMeshDevice 578	method),	<pre>set_io_sampling_rate()     (digi.xbee.devices.LPWANDevice method) 732</pre>
set_io	_configuration() (digi.xbee.devices.DigiPointDevice 610	method),	<pre>set_io_sampling_rate()     (digi.xbee.devices.NBIoTDevice</pre>
	_configuration() (digi.xbee.devices.IPDevice method), _configuration()	676	<pre>set_io_sampling_rate()      (digi.xbee.devices.Raw802Device method) 545</pre>
	(digi.xbee.devices.LPWANDevice 731	method),	<pre>set_io_sampling_rate()     (digi.xbee.devices.RemoteDigiMeshDevice</pre>
	_configuration() (digi.xbee.devices.NBIoTDevice 758	method),	<pre>method), 847 set_io_sampling_rate()    (digi.xbee.devices.RemoteDigiPointDevice)</pre>
set_io	_configuration() (digi.xbee.devices.Raw802Device 545	method),	<pre>method), 866 set_io_sampling_rate()    (digi.xbee.devices.RemoteRaw802Device)</pre>
set_io	_configuration() (digi.xbee.devices.RemoteDigiMeshD method), 847	evice	<pre>method), 828 set_io_sampling_rate()    (digi.xbee.devices.RemoteXBeeDevice)</pre>
set_io	_configuration() (digi.xbee.devices.RemoteDigiPointD method), 866	evice	<pre>method), 810 set_io_sampling_rate()    (digi.xbee.devices.RemoteZigBeeDevice</pre>
set_io	_configuration() (digi.xbee.devices.RemoteRaw802Develood), 828	vice	<pre>method), 887 set_io_sampling_rate()    (digi.xbee.devices.WiFiDevice method), 791</pre>
	_configuration() (digi.xbee.devices.RemoteXBeeDevice method), 809	2	<pre>set_io_sampling_rate()     (digi.xbee.devices.XBeeDevice</pre>
set_io	_configuration() (digi.xbee.devices.RemoteZigBeeDevi method),887	ice	<pre>set_io_sampling_rate()     (digi.xbee.devices.ZigBeeDevice method) 649</pre>
set_io	_configuration() (digi.xbee.devices.WiFiDevice method	l), 790	set_ip_address() (digi.xbee.devices.WiFiDevice method), 767
set_io	_configuration() (digi.xbee.devices.XBeeDevice 515	method),	<pre>set_ip_addressing_mode()             (digi.xbee.devices.WiFiDevice method), 766 set_local_xbee_device()</pre>
set_io	_configuration() (digi.xbee.devices.ZigBeeDevice 649	method),	(digi.xbee.devices.RemoteDigiMeshDevice method), 848 set_local_xbee_device()
set_io	_sampling_rate() (digi.xbee.devices.AbstractXBeeDevic	ce	(digi.xbee.devices.RemoteDigiPointDevice method), 867
set_io	<pre>method), 483 _sampling_rate()</pre>		<pre>set_local_xbee_device()     (digi.xbee.devices.RemoteRaw802Device</pre>

method), 829	method), 797
set_local_xbee_device()	set_ota_max_block_size()
(digi.x bee. devices. Remote XB ee Device	(digi.x bee. devices. Remote Zig Bee Device
method), 796	method), 888
set_local_xbee_device()	<pre>set_pan_id() (digi.xbee.devices.AbstractXBeeDevice</pre>
(digi.xbee.devices.RemoteZigBeeDevice	method), 480
method), 887	set_pan_id() (digi.xbee.devices.CellularDevice
<pre>set_many_to_one_broadcasting_time()</pre>	method), 705
(digi.xbee.devices.ZigBeeDevice method), 617	set_pan_id() (digi.xbee.devices.DigiMeshDevice
<pre>set_mask_address()</pre>	method), 579
(digi.xbee.devices.WiFiDevice method), 767	set_pan_id() (digi.xbee.devices.DigiPointDevice
set_node_id() (digi.xbee.devices.AbstractXBeeDevice	method), 611
method), 477	<pre>set_pan_id() (digi.xbee.devices.IPDevice method), 657</pre>
set_node_id() (digi.xbee.devices.CellularDevice	
<pre>method), 684 set_node_id() (digi.xbee.devices.DigiMeshDevice</pre>	set_pan_id() (digi.xbee.devices.LPWANDevice
set_node_id() (digi.xbee.devices.DigiMeshDevice method), 579	method), 732 set_pan_id() (digi.xbee.devices.NBIoTDevice
set_node_id() (digi.xbee.devices.DigiPointDevice	method), 758
method), 611	set_pan_id() (digi.xbee.devices.Raw802Device
set_node_id() (digi.xbee.devices.IPDevice method),	method), 546
677	set_pan_id() (digi.xbee.devices.RemoteDigiMeshDevice
<pre>set_node_id() (digi.xbee.devices.LPWANDevice</pre>	method), 848
method), 732	<pre>set_pan_id() (digi.xbee.devices.RemoteDigiPointDevice</pre>
<pre>set_node_id() (digi.xbee.devices.NBIoTDevice</pre>	method), 867
method), 758	<pre>set_pan_id() (digi.xbee.devices.RemoteRaw802Device</pre>
<pre>set_node_id() (digi.xbee.devices.Raw802Device</pre>	method), 829
method), 546	<pre>set_pan_id() (digi.xbee.devices.RemoteXBeeDevice</pre>
$\verb set_node_id()  \textit{ (digi.xbee.devices.RemoteDigiMeshDevices)} $	rice method), 810
method), 848	<pre>set_pan_id() (digi.xbee.devices.RemoteZigBeeDevice</pre>
$\verb set_node_id()  (digi.xbee.devices.RemoteDigiPointDevices.RemoteDevice$	
method), 867	set_pan_id() (digi.xbee.devices.WiFiDevice
<pre>set_node_id() (digi.xbee.devices.RemoteRaw802Devices.</pre>	
method), 829	set_pan_id() (digi.xbee.devices.XBeeDevice
<pre>set_node_id() (digi.xbee.devices.RemoteXBeeDevice</pre>	method), 516
method), 810	set_pan_id() (digi.xbee.devices.ZigBeeDevice
set_node_id() (digi.xbee.devices.RemoteZigBeeDevice	
method), 888	set_parameter() (digi.xbee.devices.AbstractXBeeDevice
set_node_id() (digi.xbee.devices.WiFiDevice	method), 474
<pre>method), 791 set_node_id()</pre>	<pre>set_parameter() (digi.xbee.devices.CellularDevice     method), 705</pre>
method), 516	set_parameter() (digi.xbee.devices.DigiMeshDevice
set_node_id() (digi.xbee.devices.ZigBeeDevice	method), 579
method), 649	set_parameter() (digi.xbee.devices.DigiPointDevice
set_ota_max_block_size()	method), 612
(digi.xbee.devices.RemoteDigiMeshDevice	set_parameter() (digi.xbee.devices.IPDevice
method), 848	method), 677
set_ota_max_block_size()	set_parameter() (digi.xbee.devices.LPWANDevice
(digi.xbee.devices.RemoteDigiPointDevice	method), 732
method), 867	<pre>set_parameter() (digi.xbee.devices.NBIoTDevice</pre>
set_ota_max_block_size()	method), 758
(digi.x bee. devices. Remote Raw 802 Device	set_parameter() (digi.xbee.devices.Raw802Device
method), 829	method), 546
set_ota_max_block_size()	$\verb set_parameter()  \textit{(digi.xbee.devices.RemoteDigiMeshDevice} \\$
(digi.xbee.devices.RemoteXBeeDevice	method), 849

```
set_parameter() (digi.xbee.devices.RemoteDigiPointDevice_power_level() (digi.xbee.devices.XBeeDevice
        method), 868
                                                            method), 516
set_parameter() (digi.xbee.devices.RemoteRaw802Deviet_power_level()
                                                            (digi.xbee.devices.ZigBeeDevice
        method), 829
                                                                                             method),
set_parameter() (digi.xbee.devices.RemoteXBeeDevice
        method), 795
                                                   set_pwm_duty_cycle()
set_parameter() (digi.xbee.devices.RemoteZigBeeDevice
                                                            (digi.xbee.devices.AbstractXBeeDevice
        method), 888
                                                            method), 484
set_parameter()
                       (digi.xbee.devices.WiFiDevice
                                                   set_pwm_duty_cycle()
        method), 792
                                                            (digi.xbee.devices.CellularDevice
                                                                                             method),
set_parameter()
                      (digi.xbee.devices.XBeeDevice
                                                   set_pwm_duty_cycle()
        method), 493
                                                            (digi.xbee.devices.DigiMeshDevice
set_parameter()
                     (digi.xbee.devices.ZigBeeDevice
                                                                                             method),
                                                            580
        method), 650
set_power_level()
                                                   set_pwm_duty_cycle()
        (digi.xbee.devices.AbstractXBeeDevice
                                                            (digi.xbee.devices.DigiPointDevice
                                                                                             method),
        method), 481
                                                            612
set_power_level()
                                                   set_pwm_duty_cycle()
        (digi.xbee.devices.CellularDevice
                                         method).
                                                            (digi.xbee.devices.IPDevice method), 678
                                                   set pwm duty cycle()
set_power_level()
                                                            (digi.xbee.devices.LPWANDevice
                                                                                             method),
        (digi.xbee.devices.DigiMeshDevice
                                         method),
        580
                                                   set_pwm_duty_cycle()
set_power_level()
                                                            (digi.xbee.devices.NBIoTDevice
                                                                                             method).
                                                            759
        (digi.xbee.devices.DigiPointDevice
                                         method),
                                                   set_pwm_duty_cycle()
set_power_level()
                         (digi.xbee.devices.IPDevice
                                                            (digi.xbee.devices.Raw802Device
                                                                                             method),
        method), 678
set_power_level()
                                                   set_pwm_duty_cycle()
        (digi.xbee.devices.LPWANDevice
                                                            (digi.xbee.devices.RemoteDigiMeshDevice
                                         method),
        732
                                                            method), 849
set_power_level()
                                                   set_pwm_duty_cycle()
        (digi.xbee.devices.NBIoTDevice
                                                            (digi.xbee.devices.RemoteDigiPointDevice
                                         method),
        758
                                                            method), 868
set_power_level()
                                                   set pwm duty cycle()
        (digi.xbee.devices.Raw802Device
                                         method),
                                                            (digi.xbee.devices.RemoteRaw802Device
        547
                                                            method), 830
set_power_level()
                                                   set_pwm_duty_cycle()
        (digi.xbee.devices.RemoteDigiMeshDevice
                                                            (digi.xbee.devices.RemoteXBeeDevice
        method), 849
                                                            method), 811
set_power_level()
                                                   set_pwm_duty_cycle()
        (digi.xbee.devices.RemoteDigiPointDevice
                                                            (digi.xbee.devices.RemoteZigBeeDevice
                                                            method), 889
        method), 868
set_power_level()
                                                   set_pwm_duty_cycle()
        (digi.xbee.devices.RemoteRaw802Device
                                                            (digi.xbee.devices.WiFiDevice method), 792
        method), 830
                                                   set_pwm_duty_cycle()
                                                            (digi.xbee.devices.XBeeDevice
set_power_level()
                                                                                             method),
        (digi.xbee.devices.RemoteXBeeDevice
                                                            517
        method), 811
                                                   set_pwm_duty_cycle()
set_power_level()
                                                            (digi.xbee.devices.ZigBeeDevice
                                                                                             method),
        (digi.xbee.devices.RemoteZigBeeDevice
                                                            651
        method), 888
                                                   set read timeout()
set_power_level() (digi.xbee.devices.WiFiDevice
                                                            (digi.xbee.serial.XBeeSerialPort
                                                                                             method),
        method), 792
                                                            1032
```

<pre>set_sync_ops_timeout()     (digi.xbee.devices.AbstractXBeeDevice</pre>	size (digi.xbee.filesystem.FileSystemElement attribute),
method), 479	size(digi.xbee.models.filesystem.OpenFileCmdResponse
set_sync_ops_timeout()	attribute), 145
(digi.xbee.devices.CellularDevice method),	size (digi.xbee.models.filesystem.ReadFileCmdRequest
(aigi.xbee.devices.CeituarDevice meinoa), 706	attribute), 150
<pre>set_sync_ops_timeout()</pre>	size_pretty (digi.xbee.filesystem.FileSystemElement
(digi.xbee.devices.DigiMeshDevice method),	attribute), 959
580	SMSMessage (class in digi.xbee.models.message), 207
<pre>set_sync_ops_timeout()</pre>	SMSReceived (class in digi.xbee.reader), 1004
(digi.xbee.devices.DigiPointDevice method),	socket (class in digi.xbee.xsocket), 1034
613	socket_id (digi.xbee.models.info.SocketInfo at-
set_sync_ops_timeout()	tribute), 196
(digi.xbee.devices.IPDevice method), 678	<pre>socket_id(digi.xbee.packets.socket.SocketBindListenPacket</pre>
<pre>set_sync_ops_timeout()</pre>	attribute), 414
(digi.xbee.devices.LPWANDevice method),	<pre>socket_id(digi.xbee.packets.socket.SocketClosePacket</pre>
732	attribute), 401
<pre>set_sync_ops_timeout()</pre>	<pre>socket_id(digi.xbee.packets.socket.SocketCloseResponsePacket</pre>
(digi.xbee.devices.NBIoTDevice method),	attribute), 404
759	<pre>socket_id(digi.xbee.packets.socket.SocketConnectPacket</pre>
<pre>set_sync_ops_timeout()</pre>	attribute), 395
(digi.xbee.devices.Raw802Device method),	<pre>socket_id(digi.xbee.packets.socket.SocketConnectResponsePacket</pre>
547	attribute), 398
set_sync_ops_timeout()	socket_id(digi.xbee.packets.socket.SocketCreateResponsePacket
(digi.xbee.devices.RemoteDigiMeshDevice	attribute), 384
method), 850	
	socket_id(digi.xbee.packets.socket.SocketListenResponsePacket
set_sync_ops_timeout()	attribute), 417
(digi.xbee.devices.RemoteDigiPointDevice	socket_id(digi.xbee.packets.socket.SocketNewIPv4ClientPacket
method), 869	attribute), 420
<pre>set_sync_ops_timeout()</pre>	socket_id(digi.xbee.packets.socket.SocketOptionRequestPacket
(digi.xbee.devices.RemoteRaw802Device	attribute), 387
method), 830	<pre>socket_id(digi.xbee.packets.socket.SocketOptionResponsePacket</pre>
<pre>set_sync_ops_timeout()</pre>	attribute), 391
$(digi.xbee.devices.RemoteXBeeDevice) \label{linear}$	<pre>socket_id(digi.xbee.packets.socket.SocketReceiveFromPacket</pre>
method), 812	attribute), 428
<pre>set_sync_ops_timeout()</pre>	<pre>socket_id(digi.xbee.packets.socket.SocketReceivePacket</pre>
(digi.xbee.devices.RemoteZigBeeDevice	attribute), 425
method), 889	<pre>socket_id (digi.xbee.packets.socket.SocketSendPacket</pre>
set_sync_ops_timeout()	attribute), 407
(digi.xbee.devices.WiFiDevice method), 792	socket_id(digi.xbee.packets.socket.SocketSendToPacket
set_sync_ops_timeout()	attribute), 410
(digi.xbee.devices.XBeeDevice method),	socket_id (digi.xbee.packets.socket.SocketStatePacket
517	attribute), 431
set_sync_ops_timeout()	SocketBindListenPacket (class in
(digi.xbee.devices.ZigBeeDevice method),	digi.xbee.packets.socket), 412
651	
	SocketClosePacket (class in
setblocking() (digi.xbee.xsocket.socket method),	digi.xbee.packets.socket), 400
1035	SocketCloseResponsePacket (class in
setsocketopt() (digi.xbee.xsocket.socket method),	digi.xbee.packets.socket), 403
1037	SocketConnectPacket (class in
settimeout() (digi.xbee.xsocket.socket method),	digi.xbee.packets.socket), 393
1035	SocketConnectResponsePacket (class in
signal_quality(digi.xbee.models.accesspoint.Access	
attribute), 130	SocketCreatePacket (class in

digi.xbee.packets.socket), 380	method), 1008
	sort() (digi.xbee.reader.ModemStatusReceived
digi.xbee.packets.socket), 382	method), 999
SocketDataReceived (class in digi.xbee.reader), 1009	sort() (digi.xbee.reader.NetworkModified method), 1001
SocketDataReceivedFrom (class in digi.xbee.reader), 1009	sort() (digi.xbee.reader.NetworkUpdateProgress method), 1017
SocketInfo (class in digi.xbee.models.info), 195	sort () (digi.xbee.reader.PacketReceived method), 996
SocketInfoState (class in digi.xbee.models.status), 230	sort() (digi.xbee.reader.PacketReceivedFrom method), 997
	<pre>sort() (digi.xbee.reader.RelayDataReceived method), 1006</pre>
SocketNewIPv4ClientPacket (class in digi.xbee.packets.socket), 418	sort() (digi.xbee.reader.RouteInformationReceived method), 1012
SocketOption (class in digi.xbee.models.options),	sort () (digi.xbee.reader.RouteReceived method), 1013
214	sort() (digi.xbee.reader.RouteRecordIndicatorReceived
SocketOptionRequestPacket (class in	method), 1011
digi.xbee.packets.socket), 386	sort () (digi.xbee.reader.SMSReceived method), 1005
digi.xbee.packets.socket), 389	sort() (digi.xbee.reader.SocketDataReceived method), 1009
SocketReceiveFromPacket (class in digi.xbee.packets.socket), 425	sort() (digi.xbee.reader.SocketDataReceivedFrom method), 1010
SocketReceivePacket (class in digi.xbee.packets.socket), 422	sort() (digi.xbee.reader.SocketStateReceived method), 1009
SocketSendPacket (class in	sort() (digi.xbee.reader.XBeeEvent method), 995
digi.xbee.packets.socket), 406 SocketSendToPacket (class in	source_address (digi.xbee.packets.network.RXIPv4Packet attribute), 343
digi.xbee.packets.socket), 409	$\verb source_address  \textit{(digi.xbee.packets.socket.SocketReceiveFromPacket} $
SocketState (class in digi.xbee.models.status), 229	attribute), 428
	·
SocketStatePacket (class in digi.xbee.packets.socket), 428	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifilattribute), 433
·	$\verb source_address  \textit{(digi.xbee.packets.wifi.IODataSampleRxIndicatorWifilal)} \\$
<pre>digi.xbee.packets.socket), 428 SocketStateReceived (class in digi.xbee.reader),</pre>	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifilattribute), 433 source_address (digi.xbee.packets.wifi.RemoteATCommandResponseV
digi.xbee.packets.socket), 428 SocketStateReceived (class in digi.xbee.reader), 1008 SocketStatus (class in digi.xbee.models.status), 229 sort() (digi.xbee.reader.BluetoothDataReceived method), 1007	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifilattribute), 433 source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442 source_endpoint (digi.xbee.models.message.ExplicitXBeeMessageattribute), 205 source_endpoint (digi.xbee.packets.common.ExplicitAddressingPackets.common.ExplicitAddressi
digi.xbee.packets.socket), 428 SocketStateReceived (class in digi.xbee.reader), 1008 SocketStatus (class in digi.xbee.models.status), 229 sort() (digi.xbee.reader.BluetoothDataReceived	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifi.attribute), 433 source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442 source_endpoint (digi.xbee.models.message.ExplicitXBeeMessageattribute), 205 source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketattribute), 300 source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPacketattribute)
digi.xbee.packets.socket), 428 SocketStateReceived (class in digi.xbee.reader), 1008 SocketStatus (class in digi.xbee.models.status), 229 sort() (digi.xbee.reader.BluetoothDataReceived method), 1007 sort() (digi.xbee.reader.DataReceived method), 998 sort() (digi.xbee.reader.DeviceDiscovered method), 1002	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifile attribute), 433 source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442 source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205 source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketattribute), 300 source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPacketattribute), 304
digi.xbee.packets.socket), 428 SocketStateReceived (class in digi.xbee.reader), 1008 SocketStatus (class in digi.xbee.models.status), 229 sort() (digi.xbee.reader.BluetoothDataReceived method), 1007 sort() (digi.xbee.reader.DataReceived method), 998 sort() (digi.xbee.reader.DeviceDiscovered method),	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifilattribute), 433 source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442 source_endpoint (digi.xbee.models.message.ExplicitXBeeMessageattribute), 205 source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketattribute), 300 source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPacketattribute)
digi.xbee.packets.socket), 428  SocketStateReceived (class in digi.xbee.reader), 1008  SocketStatus (class in digi.xbee.models.status), 229  sort () (digi.xbee.reader.BluetoothDataReceived method), 1007  sort () (digi.xbee.reader.DataReceived method), 998  sort () (digi.xbee.reader.DeviceDiscovered method), 1002  sort () (digi.xbee.reader.DiscoveryProcessFinished)	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifile attribute), 433 source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442 source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205 source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacke attribute), 300 source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPack attribute), 304 source_port (digi.xbee.models.message.IPMessage
digi.xbee.packets.socket), 428  SocketStateReceived (class in digi.xbee.reader), 1008  SocketStatus (class in digi.xbee.models.status), 229  sort () (digi.xbee.reader.BluetoothDataReceived method), 1007  sort () (digi.xbee.reader.DataReceived method), 998  sort () (digi.xbee.reader.DeviceDiscovered method), 1002  sort () (digi.xbee.reader.DiscoveryProcessFinished method), 1003  sort () (digi.xbee.reader.EndDiscoveryScan method), 1015  sort () (digi.xbee.reader.ExplicitDataReceived	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifi.attribute), 433  source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442  source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205  source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketattribute), 300  source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPackattribute), 304  source_port (digi.xbee.models.message.IPMessage attribute), 206  source_port (digi.xbee.packets.network.RXIPv4Packetattribute), 344  source_port (digi.xbee.packets.network.TXIPv4Packet
digi.xbee.packets.socket), 428  SocketStateReceived (class in digi.xbee.reader), 1008  SocketStatus (class in digi.xbee.models.status), 229  sort () (digi.xbee.reader.BluetoothDataReceived method), 1007  sort () (digi.xbee.reader.DataReceived method), 998  sort () (digi.xbee.reader.DeviceDiscovered method), 1002  sort () (digi.xbee.reader.DiscoveryProcessFinished method), 1003  sort () (digi.xbee.reader.EndDiscoveryScan method), 1015  sort () (digi.xbee.reader.ExplicitDataReceived method), 1003  sort () (digi.xbee.reader.FileSystemFrameReceived digi.xbee.reader.FileSystemFrameReceived	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifile attribute), 433  source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442  source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205  source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacke attribute), 300  source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPack attribute), 304  source_port (digi.xbee.models.message.IPMessage attribute), 206  source_port (digi.xbee.packets.network.RXIPv4Packet attribute), 344  source_port (digi.xbee.packets.network.TXIPv4Packet attribute), 349  source_port (digi.xbee.packets.socket.SocketBindListenPacket
digi.xbee.packets.socket), 428  SocketStateReceived (class in digi.xbee.reader), 1008  SocketStatus (class in digi.xbee.models.status), 229  sort () (digi.xbee.reader.BluetoothDataReceived method), 1007  sort () (digi.xbee.reader.DataReceived method), 998  sort () (digi.xbee.reader.DeviceDiscovered method), 1002  sort () (digi.xbee.reader.DiscoveryProcessFinished method), 1003  sort () (digi.xbee.reader.EndDiscoveryScan method), 1015  sort () (digi.xbee.reader.ExplicitDataReceived method), 1003  sort () (digi.xbee.reader.FileSystemFrameReceived method), 1016  sort () (digi.xbee.reader.InitDiscoveryScan method), sort () (digi.xbee.reader.InitDiscoveryScan method), 1016	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifi.attribute), 433 source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442 source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205 source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketottribute), 300 source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPacketottribute), 304 source_port (digi.xbee.models.message.IPMessagettribute), 206 source_port (digi.xbee.packets.network.RXIPv4Packetetottribute), 344 source_port (digi.xbee.packets.network.TXIPv4Packetetottribute), 349 source_port (digi.xbee.packets.socket.SocketBindListenPacketetottribute), 414 source_port (digi.xbee.packets.socket.SocketReceiveFromPacket
digi.xbee.packets.socket), 428  SocketStateReceived (class in digi.xbee.reader), 1008  SocketStatus (class in digi.xbee.models.status), 229  sort () (digi.xbee.reader.BluetoothDataReceived method), 1007  sort () (digi.xbee.reader.DataReceived method), 998  sort () (digi.xbee.reader.DeviceDiscovered method), 1002  sort () (digi.xbee.reader.DiscoveryProcessFinished method), 1003  sort () (digi.xbee.reader.EndDiscoveryScan method), 1015  sort () (digi.xbee.reader.ExplicitDataReceived method), 1003  sort () (digi.xbee.reader.FileSystemFrameReceived method), 1016  sort () (digi.xbee.reader.InitDiscoveryScan method), 1014  sort () (digi.xbee.reader.IoSampleReceived method),	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifi.attribute), 433  source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442  source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205  source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketattribute), 300  source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPackattribute), 304  source_port (digi.xbee.models.message.IPMessage attribute), 206  source_port (digi.xbee.packets.network.RXIPv4Packetattribute), 344  source_port (digi.xbee.packets.network.TXIPv4Packetattribute), 349  source_port (digi.xbee.packets.socket.SocketBindListenPacketattribute), 414  source_port (digi.xbee.packets.socket.SocketReceiveFromPacketattribute), 428  SPECIAL_FUNCTIONALITY (digi.xbee.io.IOMode at-
digi.xbee.packets.socket), 428  SocketStateReceived (class in digi.xbee.reader), 1008  SocketStatus (class in digi.xbee.models.status), 229  sort () (digi.xbee.reader.BluetoothDataReceived method), 1007  sort () (digi.xbee.reader.DataReceived method), 998  sort () (digi.xbee.reader.DeviceDiscovered method), 1002  sort () (digi.xbee.reader.DiscoveryProcessFinished method), 1003  sort () (digi.xbee.reader.EndDiscoveryScan method), 1015  sort () (digi.xbee.reader.ExplicitDataReceived method), 1003  sort () (digi.xbee.reader.FileSystemFrameReceived method), 1016  sort () (digi.xbee.reader.InitDiscoveryScan method), 1014  sort () (digi.xbee.reader.IoSampleReceived method), 1000	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifi.attribute), 433  source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442  source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205  source_endpoint (digi.xbee.packets.common.ExplicitAddressingPackeattribute), 300  source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPackattribute), 304  source_port (digi.xbee.models.message.IPMessageattribute), 206  source_port (digi.xbee.packets.network.RXIPv4Packetattribute), 344  source_port (digi.xbee.packets.network.TXIPv4Packetattribute), 349  source_port (digi.xbee.packets.socket.SocketBindListenPacketattribute), 414  source_port (digi.xbee.packets.socket.SocketReceiveFromPacketattribute), 428  SPECIAL_FUNCTIONALITY (digi.xbee.io.IOModeattribute), 985
digi.xbee.packets.socket), 428  SocketStateReceived (class in digi.xbee.reader), 1008  SocketStatus (class in digi.xbee.models.status), 229  sort () (digi.xbee.reader.BluetoothDataReceived method), 1007  sort () (digi.xbee.reader.DataReceived method), 998  sort () (digi.xbee.reader.DeviceDiscovered method), 1002  sort () (digi.xbee.reader.DiscoveryProcessFinished method), 1003  sort () (digi.xbee.reader.EndDiscoveryScan method), 1015  sort () (digi.xbee.reader.ExplicitDataReceived method), 1003  sort () (digi.xbee.reader.FileSystemFrameReceived method), 1016  sort () (digi.xbee.reader.InitDiscoveryScan method), 1014  sort () (digi.xbee.reader.IoSampleReceived method), 1000	source_address (digi.xbee.packets.wifi.IODataSampleRxIndicatorWifi.attribute), 433  source_address (digi.xbee.packets.wifi.RemoteATCommandResponseVattribute), 442  source_endpoint (digi.xbee.models.message.ExplicitXBeeMessage attribute), 205  source_endpoint (digi.xbee.packets.common.ExplicitAddressingPacketattribute), 300  source_endpoint (digi.xbee.packets.common.ExplicitRXIndicatorPackattribute), 304  source_port (digi.xbee.models.message.IPMessage attribute), 206  source_port (digi.xbee.packets.network.RXIPv4Packetattribute), 344  source_port (digi.xbee.packets.network.TXIPv4Packetattribute), 349  source_port (digi.xbee.packets.socket.SocketBindListenPacketattribute), 414  source_port (digi.xbee.packets.socket.SocketReceiveFromPacketattribute), 428  SPECIAL_FUNCTIONALITY (digi.xbee.io.IOMode at-

src event (digi.xbee,packets.digimesh.RouteInformationBackets) (digi.xbee.models.filesystem.DeleteCmdRequest attribute), 326 attribute), 183 src\_interface (digi.xbee.packets.relay.UserDataRelay@ttpttPacklegi.xbee.models.filesystem.DeleteCmdResponse attribute), 379 attribute), 184 ssid (digi.xbee.models.accesspoint.AccessPoint status (digi.xbee.models.filesystem.FileIdCmd tribute), 130 tribute), 140 (digi.xbee.reader.PacketListener (digi.xbee.models.filesystem.FileIdNameCmd start() method). status 1025 attribute), 142 status (digi.xbee.models.filesystem.FSCmd attribute), start\_discovery\_process() (digi.xbee.devices.DigiMeshNetwork method), 137 status(digi.xbee.models.filesystem.GetPathIdCmdRequeststart\_discovery\_process() attribute), 176 (digi.xbee.devices.DigiPointNetwork method), status(digi.xbee.models.filesystem.GetPathIdCmdResponse)951 attribute), 178 start\_discovery\_process() status (digi.xbee.models.filesystem.HashFileCmdRequest(digi.xbee.devices.Raw802Network method), attribute), 158 927 status (digi.xbee.models.filesystem.HashFileCmdResponse start\_discovery\_process() attribute), 160 (digi.xbee.devices.XBeeNetwork status (digi.xbee.models.filesystem.OpenDirCmdRequest method), attribute), 165 start\_discovery\_process() status (digi.xbee.models.filesystem.OpenDirCmdResponse (digi.xbee.devices.ZigBeeNetwork method), attribute), 167 915 status (digi.xbee.models.filesystem.OpenFileCmdRequest start listening() attribute), 144 status (digi.xbee.models.filesystem.OpenFileCmdResponse (digi.xbee.devices.CellularDevice method), attribute), 146 start\_listening() (digi.xbee.devices.IPDevice status (digi.xbee.models.filesystem.ReadDirCmdRequest *method*), 655 attribute), 172 start\_listening() status (digi.xbee.models.filesystem.ReadDirCmdResponse (digi.xbee.devices.LPWANDevice method), attribute), 174 733 status (digi.xbee.models.filesystem.ReadFileCmdRequest start\_listening() attribute), 151 (digi.xbee.devices.NBIoTDevice method), status (digi.xbee.models.filesystem.ReadFileCmdResponse attribute), 153 start\_listening() (digi.xbee.devices.WiFiDevice status (digi.xbee.models.filesystem.RenameCmdRequest method), 793 attribute), 180 state (digi.xbee.models.info.SocketInfo attribute), 196 status (digi.xbee.models.filesystem.RenameCmdResponse state (digi.xbee.packets.socket.SocketStatePacket atattribute), 181 tribute), 431 (digi.xbee.models.filesystem.UnknownFSCmd status status (digi.xbee.filesystem.FileProcess attribute), 960 attribute), 139 status (digi.xbee.models.atcomm.ATCommandResponse status (digi.xbee.models.filesystem.VolFormatCmdRequest attribute), 135 attribute), 189 attribute), 169 attribute), 191 attribute), 170 attribute), 186 status (digi.xbee.models.filesystem.CloseFileCmdRequeststatus (digi.xbee.models.filesystem.VolStatCmdResponse attribute), 148 attribute), 188  $\verb|status| (digi.xbee.models.filesystem.CloseFileCmdResponse+ \verb|status| (digi.xbee.models.filesystem.WriteFileCmdRequest)| |$ attribute), 149 attribute), 155  $\verb|status| (\textit{digi.xbee.models.filesystem.CreateDirCmdRequestatus} (\textit{digi.xbee.models.filesystem.WriteFileCmdResponse})|$ attribute), 162 attribute), 157 status (digi.xbee.models.filesystem.CreateDirCmdRespons& atus (digi.xbee.models.zdo.Route attribute), 238

1124 Index

status (digi.xbee.packets.common.ATCommResponsePacket

attribute), 163

```
attribute), 267
                                                                attribute), 160
status (digi.xbee.packets.common.RemoteATCommandRespærtseBacketlue (digi.xbee.models.filesystem.OpenDirCmdRequest
         attribute), 279
                                                                attribute), 165
status (digi.xbee.packets.devicecloud.DeviceResponseStatusRtucket_value (digi.xbee.models.filesystem.OpenDirCmdResponse
         attribute), 313
                                                                attribute), 167
status (digi.xbee.packets.devicecloud.SendDataResponseBacketus value (digi.xbee.models.filesystem.OpenFileCmdRequest
                                                                attribute), 144
         attribute), 324
status (digi.xbee.packets.socket.SocketCloseResponsePacketatus_value (digi.xbee.models.filesystem.OpenFileCmdResponse
         attribute), 404
                                                                attribute), 146
status (digi.xbee.packets.socket.SocketConnectResponsePacket us_value (digi.xbee.models.filesystem.ReadDirCmdRequest
         attribute), 398
                                                                attribute), 172
status (digi.xbee.packets.socket.SocketCreateResponsePackettus_value (digi.xbee.models.filesystem.ReadDirCmdResponse
        attribute), 384
                                                                attribute), 174
status (digi.xbee.packets.socket.SocketListenResponsePacketatus_value (digi.xbee.models.filesystem.ReadFileCmdRequest
         attribute), 417
                                                                attribute), 151
status (digi.xbee.packets.socket.SocketOptionResponsePaskettus_value (digi.xbee.models.filesystem.ReadFileCmdResponse
         attribute), 391
                                                                attribute), 153
status (digi.xbee.packets.wifi.RemoteATCommandResponseWifiRacketalue (digi.xbee.models.filesystem.RenameCmdRequest
         attribute), 442
                                                                attribute), 180
status (digi.xbee.packets.zigbee.RegisterDeviceStatusPacketatus_value (digi.xbee.models.filesystem.RenameCmdResponse
        attribute), 447
                                                                attribute), 181
status_a2b (digi.xbee.devices.Connection attribute), status_value(digi.xbee.models.filesystem.UnknownFSCmd
         955
                                                                attribute), 139
status b2a (digi.xbee.devices.Connection attribute), status value(digi.xbee.models.filesystem.VolFormatCmdRequest
         955
                                                                attribute), 189
status_value(digi.xbee.models.filesystem.CloseDirCmdReaguest_value(digi.xbee.models.filesystem.VolFormatCmdResponse
         attribute), 169
                                                                attribute), 191
status_value(digi.xbee.models.filesystem.CloseDirCmdResponsevalue(digi.xbee.models.filesystem.VolStatCmdRequest
        attribute), 170
                                                                attribute), 186
status_value(digi.xbee.models.filesystem.CloseFileCmdRequest_value(digi.xbee.models.filesystem.VolStatCmdResponse
         attribute), 148
                                                                attribute), 188
status_value(digi.xbee.models.filesystem.CloseFileCmdRestpons_evalue(digi.xbee.models.filesystem.WriteFileCmdRequest
        attribute), 149
                                                                attribute), 155
status_value(digi.xbee.models.filesystem.CreateDirCrustRecquest_value(digi.xbee.models.filesystem.WriteFileCmdResponse
         attribute), 162
                                                                attribute), 157
status\_value(digi.xbee.models.filesystem.CreateDirCmdResptonsdigi.xbee.models.zdo.NeighborFinder method),
        attribute), 164
status_value(digi.xbee.models.filesystem.DeleteCmdResquest()
                                                                  (digi.xbee.models.zdo.NeighborTableReader
         attribute), 183
                                                                method), 239
status_value(digi.xbee.models.filesystem.DeleteCmdRespopse)
                                                                 (digi.xbee.models.zdo.NodeDescriptorReader
                                                                method), 233
        attribute), 185
status_value(digi.xbee.models.filesystem.FileIdCmd stop()
                                                                      (digi.xbee.models.zdo.RouteTableReader
        attribute), 141
                                                                method), 237
status_value(digi.xbee.models.filesystem.FileIdNameCsrabp() (digi.xbee.reader.PacketListener method), 1017
        attribute), 142
                                                       stop_bits (digi.xbee.profile.FirmwareStopbits
status_value
                   (digi.xbee.models.filesystem.FSCmd
                                                                tribute), 987
        attribute), 137
                                                       stop_discovery_process()
status_value(digi.xbee.models.filesystem.GetPathIdCmdRequest(digi.xbee.devices.DigiMeshNetwork method),
        attribute), 176
status_value(digi.xbee.models.filesystem.GetPathIdCmdResportisescovery_process()
                                                                (digi.xbee.devices.DigiPointNetwork method),
        attribute), 178
status value (digi.xbee.models.filesystem.HashFileCmdRequest 952
         attribute), 159
                                                       stop discovery process()
status value(digi.xbee.models.filesystem.HashFileCmdResponse(digi.xbee.devices.Raw802Network
                                                                                                   method),
```

928	timeout (digi.xbee.serial.XBeeSerialPort attribute),
stop_discovery_process()	1033
(digi.xbee.devices.XBeeNetwork method), 893	TIMEOUT_READ_PACKET (digi.xbee.devices.XBeeDevice attribute),
stop_discovery_process()	(digi.xbee.devices.XBeeDevice attribute), 492
(digi.xbee.devices.ZigBeeNetwork method),	TimeoutException, 957
916	$\verb timestamp  (digi.x bee.models.message. Explicit X BeeMessage$
stop_listening() (digi.xbee.devices.CellularDevice method), 706	attribute), 205
stop_listening() (digi.xbee.devices.IPDevice	timestamp (digi.xbee.models.message.XBeeMessage attribute), 204
method), 655	timestamp (digi.xbee.packets.digimesh.RouteInformationPacke
stop_listening()(digi.xbee.devices.LPWANDevice	attribute), 326
method), 733	to_dict() (digi.xbee.models.filesystem.CloseDirCmdRequest
stop_listening() (digi.xbee.devices.NBIoTDevice	method), 169
method), 759	to_dict() (digi.xbee.models.filesystem.CloseDirCmdResponse
stop_listening() (digi.xbee.devices.WiFiDevice	method), 170
method), 793	<pre>to_dict() (digi.xbee.models.filesystem.CloseFileCmdRequest</pre>
successor_addr(digi.xbee.packets.digimesh.RouteInf	ormationPa <b>vket</b> hod), 148
attribute), 329	to_dict() (digi.xbee.models.filesystem.CloseFileCmdResponse
supports_apply_profile()	method), 149
(digi.xbee.comm_interface.XBeeCommunication	$Inter\underline{fa} cect () \ (digi.xbee.models.filesystem.CreateDirCmdRequest$
method), 472	method), 162
supports_apply_profile()	to_dict() (digi.xbee.models.filesystem.CreateDirCmdRespons
(digi.xbee.serial.XBeeSerialPort method),	method), 164
1033	to_dict() (digi.xbee.models.filesystem.DeleteCmdRequest
<pre>supports_update_firmware()</pre>	method), 183
	Interface () (digi.xbee.models.filesystem.DeleteCmdResponse
method), 471	method), 185
supports_update_firmware()	to_dict() (digi.xbee.models.filesystem.FileIdCmd
(digi.xbee.serial.XBeeSerialPort method),	method), 141
1033	to_dict() (digi.xbee.models.filesystem.FileIdNameCmd
sync_sleep (digi.xbee.firmware.UpdateConfigurer at- tribute), 976	method), 143
SyncRequestSender (class in digi.xbee.sender),	to_dict() (digi.xbee.models.filesystem.FSCmd method), 138
1029	to_dict() (digi.xbee.models.filesystem.GetPathIdCmdRequest
102)	method), 176
T	to_dict() (digi.xbee.models.filesystem.GetPathIdCmdRespons
tag (digi.xbee.profile.XBeeSettingFormat attribute), 988	method), 178
tag (digi.xbee.profile.XBeeSettingType attribute), 988	to_dict() (digi.xbee.models.filesystem.HashFileCmdRequest
target (digi.xbee.packets.devicecloud.DeviceRequestPac	
attribute), 307	to_dict() (digi.xbee.models.filesystem.HashFileCmdResponse
task (digi.xbee.models.status.UpdateProgressStatus at-	method), 160
tribute), 232	<pre>to_dict() (digi.xbee.models.filesystem.OpenDirCmdRequest</pre>
task_done() (digi.xbee.reader.XBeeQueue method),	method), 165
1028	$\verb"to_dict()" (\textit{digi.xbee.models.filesystem.OpenDirCmdResponse"}$
timeout (digi.xbee.comm_interface.XBeeCommunication	nInterface method), 167
attribute), 473	$\verb"to_dict()" (\textit{digi.xbee.models.filesystem.OpenFileCmdRequest"}) \\$
timeout (digi.xbee.firmware.FwUpdateTask attribute),	method), 145
978	to_dict() (digi.xbee.models.filesystem.OpenFileCmdResponse
timeout (digi.xbee.profile.ProfileUpdateTask at-	method), 146
tribute), 994	to_dict() (digi.xbee.models.filesystem.ReadDirCmdRequest
timeout (digi.xbee.sender.SyncRequestSender at-	method), 172
tribute), 1030	to_dict() (digi.xbee.models.filesystem.ReadDirCmdResponse
	method), 174

- to\_dict() (digi.xbee.models.filesystem.ReadFileCmdRequestdict() (digi.xbee.packets.common.IODataSampleRxIndicatorPacket method), 151 method), 296
- to\_dict() (digi.xbee.models.filesystem.ReadFileCmdResp@nseict() (digi.xbee.packets.common.ModemStatusPacket method), 153 method), 292
- to\_dict() (digi.xbee.models.filesystem.RenameCmdRequæst\_dict() (digi.xbee.packets.common.ReceivePacket method), 180 method), 273
- to\_dict() (digi.xbee.models.filesystem.RenameCmdResponsedict() (digi.xbee.packets.common.RemoteATCommandPacket method), 181 method), 276
- to\_dict() (digi.xbee.models.filesystem.UnknownFSCmdto\_dict() (digi.xbee.packets.common.RemoteATCommandResponsePac method), 139 method), 281
- to\_dict() (digi.xbee.models.filesystem.VolFormatCmdRequextict() (digi.xbee.packets.common.TransmitPacket method), 189 method), 285
- to\_dict() (digi.xbee.models.filesystem.VolFormatCmdResponstact() (digi.xbee.packets.common.TransmitStatusPacket method), 191 method), 289
- to\_dict() (digi.xbee.models.filesystem.VolStatCmdRequest\_dict() (digi.xbee.packets.devicecloud.DeviceRequestPacket method), 186 method), 308
- to\_dict() (digi.xbee.models.filesystem.VolStatCmdResponse\_dict() (digi.xbee.packets.devicecloud.DeviceResponsePacket method), 188 method), 311
- to\_dict() (digi.xbee.models.filesystem.WriteFileCmdRequæstdict() (digi.xbee.packets.devicecloud.DeviceResponseStatusPacket method), 155 method), 314
- to\_dict() (digi.xbee.models.filesystem.WriteFileCmdResponxwict() (digi.xbee.packets.devicecloud.FrameErrorPacket method), 157 method), 317
- to\_dict() (digi.xbee.models.message.ExplicitXBeeMessage\_dict() (digi.xbee.packets.devicecloud.SendDataRequestPacket method), 205 method), 321
- to\_dict() (digi.xbee.models.message.IPMessage to\_dict() (digi.xbee.packets.devicecloud.SendDataResponsePacket method), 207 method), 323
- to\_dict() (digi.xbee.models.message.SMSMessage to\_dict() (digi.xbee.packets.digimesh.RouteInformationPacket method), 207 method), 328
- to\_dict() (digi.xbee.models.message.UserDataRelayMetsagedict() (digi.xbee.packets.filesystem.FSRequestPacket method), 208 method), 332
- to\_dict() (digi.xbee.models.message.XBeeMessage to\_dict() (digi.xbee.packets.filesystem.FSResponsePacket method), 204 method), 334
- to\_dict() (digi.xbee.packets.base.GenericXBeePacket to\_dict() (digi.xbee.packets.filesystem.RemoteFSRequestPacket method), 250 method), 338
- to\_dict() (digi.xbee.packets.base.UnknownXBeePacket to\_dict() (digi.xbee.packets.filesystem.RemoteFSResponsePacket method), 253 method), 340
- to\_dict() (digi.xbee.packets.base.XBeeAPIPacket to\_dict() (digi.xbee.packets.network.RXIPv4Packet method), 248 method), 345
- to\_dict() (digi.xbee.packets.base.XBeePacket to\_dict() (digi.xbee.packets.network.TXIPv4Packet method), 245 method), 348
- to\_dict() (digi.xbee.packets.cellular.RXSMSPacket to\_dict() (digi.xbee.packets.raw.RX16IOPacket method), 256 method), 371
- to\_dict() (digi.xbee.packets.cellular.TXSMSPacket to\_dict() (digi.xbee.packets.raw.RX16Packet method), 365
- to\_dict() (digi.xbee.packets.common.ATCommPacket to\_dict() (digi.xbee.packets.raw.RX64IOPacket method), 262 method), 369
- to\_dict() (digi.xbee.packets.common.ATCommQueuePatket\_dict() (digi.xbee.packets.raw.RX64Packet method), 362
- to\_dict() (digi.xbee.packets.common.ATCommResponsetPacketct() (digi.xbee.packets.raw.TX16Packet method), 269 method), 355
- to\_dict() (digi.xbee.packets.common.ExplicitAddressingPacketct() (digi.xbee.packets.raw.TX64Packet method), 299 method), 352
- to\_dict() (digi.xbee.packets.common.ExplicitRXIndicatorPacket() (digi.xbee.packets.raw.TXStatusPacket method), 303 method), 358

```
to_dict() (digi.xbee.packets.relay.UserDataRelayOutputPacketmit_options (digi.xbee.packets.common.RemoteATCommandPa
                  method), 379
                                                                                                                                    attribute), 275
to_dict() (digi.xbee.packets.relay.UserDataRelayPackettransmit_options(digi.xbee.packets.common.TransmitPacket
                  method), 376
                                                                                                                                    attribute), 283
to_dict() (digi.xbee.packets.socket.SocketBindListenPacketansmit_options (digi.xbee.packets.filesystem.RemoteFSRequestPackets.
                  method), 415
                                                                                                                                    attribute), 336
to_dict() (digi.xbee.packets.socket.SocketClosePacket transmit_options(digi.xbee.packets.network.TXIPv4Packet
                  method), 402
                                                                                                                                     attribute), 349
to_dict() (digi.xbee.packets.socket.SocketCloseResponseRewkermit_options (digi.xbee.packets.raw.TX16Packet
                  method), 406
                                                                                                                                    attribute), 354
\verb|to_dict()| (\textit{digi.xbee.packets.socket.SocketConnectPacket}| ransmit\_options (\textit{digi.xbee.packets.raw.TX64Packet}| ransmit\_options (\textit{digi.xbee.packets.raw.TX64Packet, ransmit\_options (\textit{digi.xbee.packets.raw.TX64
                                                                                                                                     attribute), 350
                  method), 396
to_dict() (digi.xbee.packets.socket.SocketConnectResponsæPackett_options (digi.xbee.packets.wifi.RemoteATCommandWifiPac
                  method), 399
                                                                                                                                     attribute), 437
to_dict() (digi.xbee.packets.socket.SocketCreatePacket transmit_retry_count
                  method), 382
                                                                                                                                     (digi.xbee.packets.common.TransmitStatusPacket
to_dict() (digi.xbee.packets.socket.SocketCreateResponsePacket attribute), 287
                  method), 385
                                                                                                                 transmit_status(digi.xbee.packets.common.TransmitStatusPacket
to_dict() (digi.xbee.packets.socketListenResponsePacket attribute), 287
                  method), 418
                                                                                                                 transmit_status(digi.xbee.packets.raw.TXStatusPacket
to_dict() (digi.xbee.packets.socket.SocketNewIPv4ClientPacket attribute), 357
                  method), 422
                                                                                                                 TransmitException, 957
to_dict() (digi.xbee.packets.socket.SocketOptionRequesTPacketsmitOptions
                                                                                                                                                                                      (class
                                                                                                                                                                                                                           in
                                                                                                                                     digi.xbee.models.options), 209
                  method), 389
to_dict()(digi.xbee.packets.socket.SocketOptionResponTerPanketitPacket
                                                                                                                                                                                                                           in
                  method), 392
                                                                                                                                    digi.xbee.packets.common), 281
to_dict() (digi.xbee.packets.socket.SocketReceiveFromPackets.smitStatus (class in digi.xbee.models.status),
                  method), 427
to_dict() (digi.xbee.packets.socket.SocketReceivePacketTransmitStatusPacket
                                                                                                                                                                                             (class
                                                                                                                                                                                                                           in
                  method), 424
                                                                                                                                     digi.xbee.packets.common), 285
to_dict() (digi.xbee.packets.socket.SocketSendPacket transport(digi.xbee.packets.devicecloud.DeviceRequestPacket
                  method), 409
                                                                                                                                     attribute), 307
\verb|to_dict()| (digi.xbee.packets.socket.SocketSendToPacket TRUNCATE (digi.xbee.models.options.FileOpenRequestOption)| | Truncate (digi.xbee.models.options)| | Trunca
                  method), 412
                                                                                                                                     attribute), 215
to_dict() (digi.xbee.packets.socket.SocketStatePacket TX16Packet (class in digi.xbee.packets.raw), 352
                                                                                                                 TX64Packet (class in digi.xbee.packets.raw), 349
                  method), 430
to_dict() (digi.xbee.packets.wifi.IODataSampleRxIndicaterWifiBacketount (digi.xbee.packets.digimesh.RouteInformationPacket
                  method), 435
                                                                                                                                     attribute), 326
to_dict() (digi.xbee.packets.wifi.RemoteATCommandReTYXODENWIFiRakket (class in digi.xbee.packets.network),
                  method), 441
                                                                                                                                     345
to_dict() (digi.xbee.packets.wifi.RemoteATCommandWiftRank&Packet (class in digi.xbee.packets.cellular), 256
                  method), 438
                                                                                                                  TXStatusPacket (class in digi.xbee.packets.raw),
to_dict() (digi.xbee.packets.zigbee.CreateSourceRoutePacket
                                                                                                                                     356
                                                                                                                  type (digi.xbee.models.filesystem.CloseDirCmdRequest
                  method), 457
to_dict() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacketibute), 169
                                                                                                                  type (digi.xbee.models.filesystem.CloseDirCmdResponse
                  method), 459
to_dict() (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket attribute), 171
                                                                                                                  type (digi.xbee.models.filesystem.CloseFileCmdRequest
                  method), 449
to_dict() (digi.xbee.packets.zigbee.RegisterJoiningDevicePacket attribute), 148
                                                                                                                 type(digi.xbee.models.filesystem.CloseFileCmdResponse
                  method), 446
to_dict() (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket attribute), 150
                                                                                                                  type(digi.xbee.models.filesystem.CreateDirCmdRequest
                  method), 453
transmit_options (digi.xbee.packets.common.ExplicitAddressingParithete), 162
                  attribute), 300
                                                                                                                 type (digi.xbee.models.filesystem.CreateDirCmdResponse
```

- attribute), 164
- type (digi.xbee.models.filesystem.DeleteCmdRequest at*tribute*), 183
- (digi.xbee.models.filesystem.DeleteCmdResponse type attribute), 185
- type (digi.xbee.models.filesystem.FileIdCmd attribute),
- type (digi.xbee.models.filesystem.FileIdNameCmd attribute), 143
- type (digi.xbee.models.filesystem.FSCmd attribute), 137
- type(digi.xbee.models.filesystem.GetPathIdCmdRequestattribute), 176
- attribute), 178
- type (digi.xbee.models.filesystem.HashFileCmdRequest attribute), 159
- $\verb|type| (\textit{digi.xbee.models.filesystem.HashFileCmdResponse}| \verb|unescape_data()| (\textit{digi.xbee.packets.common.ATCommPacket}| | \textit{type}(\textit{digi.xbee.models.filesystem.HashFileCmdResponse}| | \textit{unescape_data()}| | \textit{digi.xbee.packets.common.ATCommPacket}| | \textit{type}(\textit{digi.xbee.packets.common.ATCommPacket}| | \textit{type}(\textit{digi.xbee.packets.common.atcommo$ attribute), 161
- type (digi.xbee.models.filesystem.OpenDirCmdRequest attribute), 166
- type (digi.xbee.models.filesystem.OpenDirCmdResponse attribute), 168
- type (digi.xbee.models.filesystem.OpenFileCmdRequest attribute), 145
- attribute), 147
- type (digi.xbee.models.filesystem.ReadDirCmdRequest attribute), 172
- type (digi.xbee.models.filesystem.ReadDirCmdResponse attribute), 174
- type (digi.xbee.models.filesystem.ReadFileCmdRequest attribute), 152
- type(digi.xbee.models.filesystem.ReadFileCmdResponse)attribute), 153
- type (digi.xbee.models.filesystem.RenameCmdRequest attribute), 180
- type (digi.xbee.models.filesystem.RenameCmdResponse attribute), 182
- type (digi.xbee.models.filesystem.UnknownFSCmd attribute), 138
- $\verb|type| (\textit{digi.xbee.models.filesystem.VolFormatCmdRequest} \ \ \texttt{unescape\_data()} \ \ (\textit{digi.xbee.packets.devicecloud.DeviceRequestPackets}) \\$ attribute), 190
- $\verb|type| (digi.xbee.models.filesystem.VolFormatCmdResponse \verb|unescape_data|)| (digi.xbee.packets.devicecloud.DeviceResponsePackets)| (digi.xbee.packets)| (digi.xbee.packe$ attribute), 191
- (digi.xbee.models.filesystem.VolStatCmdRequesttype attribute), 186
- type (digi.xbee.models.filesystem.VolStatCmdResponse attribute), 188
- type (digi.xbee.models.filesystem.WriteFileCmdRequest attribute), 155
- attribute), 157
- type (digi.xbee.models.status.UpdateProgressStatus at- unescape\_data() (digi.xbee.packets.digimesh.RouteInformationPacket tribute), 232

(digi.xbee.profile.XBeeProfileSetting attribute), type

- unescape\_data() (digi.xbee.packets.base.GenericXBeePacket static method), 250
- unescape\_data() (digi.xbee.packets.base.UnknownXBeePacket static method), 253
- unescape\_data() (digi.xbee.packets.base.XBeeAPIPacket static method), 248
- unescape\_data() (digi.xbee.packets.base.XBeePacket static method), 245
- static method), 256
  - unescape\_data()(digi.xbee.packets.cellular.TXSMSPacket static method), 259
  - static method), 262
  - unescape\_data() (digi.xbee.packets.common.ATCommQueuePacket static method), 265
  - unescape\_data() (digi.xbee.packets.common.ATCommResponsePacke static method), 269
  - unescape data() (digi.xbee.packets.common.ExplicitAddressingPacke static method), 299
- ${\tt type} \ (\textit{digi.xbee.models.filesystem.OpenFileCmdResponse} \ \ {\tt unescape\_data()} \ (\textit{digi.xbee.packets.common.ExplicitRXIndicatorPack$ static method), 303
  - unescape\_data()(digi.xbee.packets.common.IODataSampleRxIndicat static method), 296
  - unescape\_data() (digi.xbee.packets.common.ModemStatusPacket static method), 292
  - unescape\_data() (digi.xbee.packets.common.ReceivePacket static method), 273
  - unescape\_data() (digi.xbee.packets.common.RemoteATCommandPack static method), 277
  - unescape\_data() (digi.xbee.packets.common.RemoteATCommandResp static method), 281
  - unescape\_data() (digi.xbee.packets.common.TransmitPacket static method), 285
  - unescape\_data() (digi.xbee.packets.common.TransmitStatusPacket static method), 289
  - static method), 308
  - static method), 312
  - unescape\_data() (digi.xbee.packets.devicecloud.DeviceResponseStatu static method), 315
  - unescape\_data() (digi.xbee.packets.devicecloud.FrameErrorPacket static method), 317
  - unescape\_data() (digi.xbee.packets.devicecloud.SendDataRequestPac static method), 321
- type (digi.xbee.models.filesystem.WriteFileCmdResponse unescape\_data() (digi.xbee.packets.devicecloud.SendDataResponsePackets.devicecloud. static method), 323
  - static method), 328

```
unescape_data() (digi.xbee.packets.filesystem.FSRequextPacketpe_data() (digi.xbee.packets.socket.SocketReceivePacket
                                                               static method), 424
        static method), 332
unescape_data() (digi.xbee.packets.filesystem.FSResponsePackete_data() (digi.xbee.packets.socket.SocketSendPacket
        static method), 335
                                                               static method), 409
unescape_data() (digi.xbee.packets.filesystem.RemoteESRequestPacketta() (digi.xbee.packets.socketSendToPacket
                                                               static method), 412
        static method), 338
unescape_data() (digi.xbee.packets.filesystem.RemoteESRespayseParketa() (digi.xbee.packets.socket.SocketStatePacket
         static method), 340
                                                               static method), 430
unescape_data() (digi.xbee.packets.network.RXIPv4Packetscape_data() (digi.xbee.packets.wifi.IODataSampleRxIndicatorWi
        static method), 345
                                                               static method), 435
unescape_data() (digi.xbee.packets.network.TXIPv4Packetscape_data() (digi.xbee.packets.wifi.RemoteATCommandResponse
        static method), 348
                                                               static method), 441
unescape_data() (digi.xbee.packets.raw.RX16IOPacketinescape_data() (digi.xbee.packets.wifi.RemoteATCommandWifiPacketinescape_data()
                                                               static method), 439
        static method), 371
unescape_data() (digi.xbee.packets.raw.RX16Packet unescape_data() (digi.xbee.packets.zigbee.CreateSourceRoutePacket
        static method), 366
                                                               static method), 457
unescape_data() (digi.xbee.packets.raw.RX64IOPacketinescape_data() (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatu
        static method), 369
                                                              static method), 459
unescape_data() (digi.xbee.packets.raw.RX64Packet unescape_data() (digi.xbee.packets.zigbee.RegisterDeviceStatusPacket
        static method), 362
                                                               static method), 449
unescape_data() (digi.xbee.packets.raw.TX16Packet unescape_data() (digi.xbee.packets.zigbee.RegisterJoiningDevicePack
                                                              static method), 446
        static method), 355
unescape_data() (digi.xbee.packets.raw.TX64Packet unescape_data() (digi.xbee.packets.zigbee.RouteRecordIndicatorPack
        static method), 352
                                                              static method), 453
unescape_data() (digi.xbee.packets.raw.TXStatusPacketNKNOWN (digi.xbee.devices.LinkQuality attribute), 953
                                                     \verb"UNKNOWN\_ADDRESS" (digi.xbee.models.address.XBee16BitAddress")
        static method), 358
unescape_data() (digi.xbee.packets.relay.UserDataRelayOutputPatokibtute), 200
                                                      UNKNOWN_ADDRESS (digi.xbee.models.address.XBee64BitAddress
        static method), 379
unescape_data() (digi.xbee.packets.relay.UserDataRelayPacket attribute), 202
        static method), 376
                                                      UNKNOWN_VALUE
                                                                              (digi.xbee.devices.LinkQuality
unescape_data() (digi.xbee.packets.socket.SocketBindListenPackættribute), 953
        static method), 415
                                                     UnknownFSCmd (class in digi.xbee.models.filesystem),
unescape_data() (digi.xbee.packets.socket.SocketClosePacket 138
        static method), 402
                                                     UnknownXBeePacket
                                                                                        (class
                                                                                                       in
unescape_data() (digi.xbee.packets.socket.SocketCloseResponseRigikebee.packets.base), 251
        static method), 406
                                                     unregister_joining_device()
unescape_data() (digi.xbee.packets.socket.SocketConnectPacket(digi.xbee.devices.ZigBeeDevice
                                                                                                 method),
        static method), 396
unescape_data()(digi.xbee.packets.socket.SocketConnectResponsePacketining_device_async()
        static method), 400
                                                               (digi.xbee.devices.ZigBeeDevice
                                                                                                 method),
unescape_data()(digi.xbee.packets.socket.SocketCreatePacket 624
         static method), 382
                                                      update_bluetooth_password()
unescape_data() (digi.xbee.packets.socket.SocketCreateResponseProjetee.devices.AbstractXBeeDevice
        static method), 386
                                                               method), 489
unescape_data() (digi.xbee.packets.socket.SocketListemPesperseBackettooth_password()
                                                               (digi.xbee.devices.CellularDevice
        static method), 418
                                                                                                 method),
unescape_data() (digi.xbee.packets.socket.SocketNewIPv4ClientPa6ket
        static method), 422
                                                     update_bluetooth_password()
unescape_data() (digi.xbee.packets.socket.SocketOptionRequestPdigiexbee.devices.DigiMeshDevice
                                                                                                 method),
        static method), 389
unescape_data() (digi.xbee.packets.socket.SocketOptiomRespons@Packetooth_password()
        static method), 393
                                                               (digi.xbee.devices.DigiPointDevice
                                                                                                 method),
unescape_data() (digi.xbee.packets.socket.SocketReceiveFromPacket
        static method), 427
                                                     update bluetooth password()
```

(digi.xbee.devices.LPWANDevice   733		undate	(digi.xbee.devices.IPDevice method), _bluetooth_password()	678		(digi.xbee.devices.Raw802Device 548	method),
T33	Table	ираасс		method)	undate		1
method, 850   method, 850	wethod, 850			memou),	араасс		
(digi.xbee.devices.NBIoTDevice   759		updat.e	,				
159	This is a content of the password ()	o.[- 0.0.0 c		method).	update		)
Square	wethod, 869		_	,,,,	1		
(digi.xbee.devices.Raw802Device   method), 547	digi.xbee.devices.Raw802Device	update	bluetooth password()				
1		-		method),	update		)
(digi.xbee.devices.RemoteDigiMeshDevice method), \$50			547				
method), \$50	method), 850	update	_bluetooth_password()			method), 831	
update_bluetooth_password()         (digi.xbee.devices.RemoteDigiPointDevice method), 812           update_bluetooth_password()         (digi.xbee.devices.RemoteZigBeeDevice method), 890           update_bluetooth_password()         (digi.xbee.devices.WiFiDevice method), 793           update_bluetooth_password()         (digi.xbee.devices.WiFiDevice method), 793           update_bluetooth_password()         (digi.xbee.devices.XBeeDevice method), 889           update_bluetooth_password()         (digi.xbee.devices.ZigBeeDevice method), 889           update_bluetooth_password()         (digi.xbee.devices.AbstractXBeeDevice method), 473           update_bluetooth_password()         (digi.xbee.devices.CellularDevice method), 489           update_bluetooth_salt_verifier()         (digi.xbee.devices.CellularDevice method), 489           update_bluetooth_salt_verifier()         (digi.xbee.devices.DigiNeshDevice method), 489           update_bluetooth_salt_verifier()         (digi.xbee.devices.DigiNeshDevice method), 679           update_bluetooth_salt_verifier()         (digi.xbee.devices.DigiNeshDevice method), 6	update_bluetooth_password ()		(digi.xbee.devices.RemoteDigiMeshDigiNeshDig	evice -	update	_bluetooth_salt_verifier()	)
digi.xbee.devices.RemoteDigiPointDevice	digi.xbee.devices.RemoteDigiPointDevice		method), 850			(digi.xbee.devices.RemoteXBeeDevic	e
method), 869	method, 869	update	_bluetooth_password()			method), 812	
update_bluetooth_password()         (digi.xbee.devices.RemoteRaw802Device method), 890         update_bluetooth_salt_verifier()         (digi.xbee.devices.WFiDevice method), 793           update_bluetooth_password()         (digi.xbee.devices.XBeeDevice method), 812         (digi.xbee.devices.XBeeDevice method), 818         method), 812           update_bluetooth_password()         (digi.xbee.devices.XEeDevice digi.xbee.devices.XEeDevice method), 889         (digi.xbee.devices.XEeDevice method), 652         (digi.xbee.devices.XEeDevice method), 473         (digi.xbee.devices.XEeDevice method), 473         (digi.xbee.devices.AbstractXBeeDevice method), 473         (digi.xbee.devices.AbstractXBeeDevice method), 473         method), 483         update_device_data_from ()         (digi.xbee.devices.CellularDevice method), 473         method), 483         update_device_data_from ()         (digi.xbee.devices.DigiMeshDevice method), 473         method), 483         update_device_data_from ()         (digi.xbee.devices.DigiMeshDevice method), 473         method), 483         update_device_data_from ()         (digi.xbee.devices.DigiPointDevice method), 473         method), 483         update_device_data_from ()         (digi.xbee.devices.DigiPointDevice method), 479         method), 483         update_device_data_from ()         (digi.xbee.devices.DigiPointDevice method), 479         update_device_data_from ()         (digi.xbee.devices.DigiPointDevice method), 479         update_device_data_from ()         (digi.xbee.devices.DigiPointDevice method), 479         update_device_data_from ()         (digi.xbee.devices.DigiPointD	update_bluetooth_password()		(digi.xbee.devices.RemoteDigiPointD	evice	update		
							ice
method), 830		update					
update_bluetooth_password()         (digi.xbee.devices.RBeeDevice method), 812         (digi.xbee.devices.XBeeDevice digi.xbee.devices.XBeeDevice method), 889         update_bluetooth_password()         (digi.xbee.devices.ZigBeeDevice method), 889         (digi.xbee.devices.ZigBeeDevice method), 889         652         update_bluetooth_password()         (digi.xbee.devices.ZigBeeDevice method), 8793         update_bluetooth_password()         (digi.xbee.devices.AbstractXBeeDevice method), 473         (digi.xbee.devices.AbstractXBeeDevice method), 473         update_bluetooth_password()         (digi.xbee.devices.AbstractXBeeDevice method), 473         update_devices.AbstractXBeeDevice method), 473         update_devices.AbstractXBeeDevice method), 473         update_devices.CellularDevice method), 473         method), 489         update_device_data_from()         method), 473         update_device_s.DigiPointDevice         method), 473         update_device_s.DigiPointDevice         method), 473         update_device_s.DigiPointDevice         method),	update_bluetooth_password()			vice	update		
						-	
method), 812	method), 812	update			update		
update_bluetooth_password()         (digi.xbee.devices.RemoteZigBeeDevice method), 889         (digi.xbee.devices.ZigBeeDevice method), 652         method), 889         652         update_bluetooth_password()         update_device_data_from()         (digi.xbee.devices.AbstractXBeeDevice method), 473         update_devices.AbstractXBeeDevice digi.xbee.devices.AbstractXBeeDevice (digi.xbee.devices.CellularDevice method), 473         method), 473         update_devices.CellularDevice method), 473         update_devices.CellularDevice method), 470         method), 473         update_devices.DigiMeshDevice method), 470         method), 473         update_devices.DigiMeshDevice method), 470         method), 473         update_devices.DigiMeshDevice method), 670         method), 473         update_devices.DigiMeshDevice method), 679         method), 489         update_devices.DigiPointDevice method), 679         update_devices.DigiPointDevice method), 679         update_devices.LPWaNDevice	update_bluetooth_password()         (digi.xbee.devices.RemoteZigBeeDevice method), 889         (digi.xbee.devices.ZigBeeDevice method), 890         (digi.xbee.devices.XijBeeDevice method), 473         (update_bluetooth_password()         (update_device_data_from()         (digi.xbee.devices.AbstractXBeeDevice method), 473         (update_bluetooth_password()         (update_device_data_from()         method), 473           update_bluetooth_password()         (digi.xbee.devices.CellularDevice method), 473         update_device_data_from()         method), 470           update_bluetooth_password()         (digi.xbee.devices.CellularDevice method), 470         470         method), 470           update_bluetooth_password()         (digi.xbee.devices.DigiMeshDevice method), 470         update_devices.DigiMeshDevice method), 613         method), 470           update_bluetooth_salt_verifier()         (digi.xbee.devices.DigiPointDevice method), 470         update_device_data_from()         method), 470           update_bluetooth_salt_verifier()         (digi.xbee.devices.DigiPointDevice method), 470         update_device_data_from()         method), 470           update_bluetooth_salt_verifier()         (digi.xbee.devices.IPWANDevice method), 470         update_device_data_from()         method), 470           update_bluetooth_salt_verifier()         (digi.xbee.devices.IPWANDevice method), 470         update_device_data_from()         method), 470           update_bluetooth_salt_verifier()         (digi.xbee.devices.R			e		-	method),
method), 889  update_bluetooth_password()     (digi.xbee.devices.WiFiDevice method), 793  update_bluetooth_password()     (digi.xbee.devices.XBeeDevice	method), 889  update_bluetooth_password()	update		•	update		
update_bluetooth_password()         update_device_data_from()         (digi.xbee.devices.AbstractXBeeDevice method), 473           update_bluetooth_password()         method), 473         update_device_data_from()         method), 473           update_bluetooth_password()         (digi.xbee.devices.CellularDevice method), 470         method), 473         method), 473           update_bluetooth_password()         (digi.xbee.devices.CellularDevice method), 470         707         method), 480         method), 489         update_device_data_from()         method), 619           update_bluetooth_salt_verifier()         method), 480         update_device_data_from()         method), 619           update_bluetooth_salt_verifier()         method), 613         update_device_data_from()         method), 679           update_bluetooth_salt_verifier()         method), 679         update_device_data_from()         method), 679           update_bluetooth_salt_verifier()         method), 670         (digi.xbee.devices.IPWANDevice method), 679         method), 670           update_bluetooth_salt_verifier()         method), 670         (digi.xbee.device_data_from()         method), 670           update_bluetooth_salt_verifier()         digi.xbee.device_data_from()         (digi.xbee.device_s.Raw802Device method), 670	<pre>update_bluetooth_password()    (digi.xbee.devices.WiFiDevice method), 793 update_bluetooth_password()    (digi.xbee.devices.XBeeDevice</pre>		-	ice			тетоа),
		undata			undata		
update_bluetooth_password()         method), 473           (digi.xbee.devices.XBeeDevice 517         method), 473           update_bluetooth_password()         (digi.xbee.devices.CellularDevice 707           (digi.xbee.devices.ZigBeeDevice 651         method), 489           (digi.xbee.devices.AbstractXBeeDevice method), 489         (digi.xbee.devices.DigiMeshDevice 707         method), 489           update_bluetooth_salt_verifier()         method), 489         update_device_data_from() (digi.xbee.devices.DigiPointDevice 707         method), 489           update_bluetooth_salt_verifier()         method), 489         update_device_data_from() (digi.xbee.devices.DigiPointDevice 707         method), 489           update_bluetooth_salt_verifier()         method), 480         update_device_data_from() (digi.xbee.devices.DigiPointDevice 707         method), 480           update_bluetooth_salt_verifier()         digis.xbee.devices.IPDevice method), 679         update_device_data_from() (digi.xbee.devices.DigiPointDevice 734         method), 734           update_bluetooth_salt_verifier()         method), 613         update_device_data_from() (digi.xbee.devices.NBloTDevice 760         method), 760           update_bluetooth_salt_verifier()         digi.xbee.devices.RemoteDigiMeshDevice 733         method), 851           update_bluetooth_salt_verifier()         digi.xbee.devices.RemoteDigiPointDevice 733         method), 851           update_bluetooth_salt_verifier()	update_bluetooth_password()	upuate		d) 793	upuate		c e
(digi.xbee.devices.XBeeDevice   method),		undate	_	1), 1)3		-	
		араасс		method).	update		
<pre>update_bluetooth_password()     (digi.xbee.devices.ZigBeeDevice</pre>	update_bluetooth_password()		_	,	ap aacc		method).
digi.xbee.devices.ZigBeeDevice	(digi.xbee.devices.ZigBeeDevicemethod),update_device_data_from()method),update_bluetooth_salt_verifier()581update_devices.AbstractXBeeDeviceupdate_device_data_from()method),update_bluetooth_salt_verifier()614(digi.xbee.devices.CellularDevicemethod),614\(\frac{1}{2}\) (digi.xbee.devices.CellularDevicemethod),update_device_data_from()679update_bluetooth_salt_verifier()digi.xbee.devices.IPDevice method),679\(\frac{1}{2}\) (digi.xbee.devices.DigiMeshDevice}method),digi.xbee.devices.LPWANDevicemethod),\(\frac{1}{2}\) (digi.xbee.devices.DigiPointDevice}method),\(\frac{1}{2}\) (digi.xbee.devices.NBIoTDevicemethod),\(\frac{1}{2}\) (digi.xbee.devices.IPDevice method)679\(\frac{1}{2}\) (digi.xbee.devices.Raw802Devicemethod),\(\frac{1}{2}\) (digi.xbee.devices.Raw802Devicemethod),548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiMeshDevicemethod),\$548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiMeshDevicemethod),\$548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiMeshDevicemethod),\$548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiMeshDevicemethod),\$548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiMeshDevicemethod),\$548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiMeshDevicemethod),\$548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiMeshDevicemethod),\$548\(\frac{1}{2}\) (digi.xbee.devices.RemoteDigiPointDevicemethod),\$649\(\frac{1}{2}\) (d	update	bluetooth password()				,,
		-		method),	update	_device_data_from()	
<pre>(digi.xbee.devices.AbstractXBeeDevice     method), 489  update_bluetooth_salt_verifier()     (digi.xbee.devices.CellularDevice</pre>	<pre>(digi.xbee.devices.AbstractXBeeDevice     method), 489  update_bluetooth_salt_verifier()</pre>			,	-		method),
method), 489(digi.xbee.devices.DigiPointDevicemethod),update_bluetooth_salt_verifier()method),update_device_data_from()707(digi.xbee.devices.IPDevice method),(digi.xbee.devices.IPDevice method),update_bluetooth_salt_verifier()method),(digi.xbee.devices.LPWANDevicemethod),581734update_bluetooth_salt_verifier()update_device_data_from()method),(digi.xbee.devices.DigiPointDevicemethod),(digi.xbee.devices.NBIoTDevicemethod),613760update_device_data_from()method),update_bluetooth_salt_verifier()update_device_data_from()method),(digi.xbee.devices.IPDevice method), 679(digi.xbee.devices.Raw802Devicemethod),update_bluetooth_salt_verifier()548(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method),(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method),851(digi.xbee.devices.NBIoTDevicemethod),update_device_data_from()(digi.xbee.devices.RemoteDigiPointDevicemethod),	<pre>method), 489 update_bluetooth_salt_verifier() (digi.xbee.devices.CellularDevice</pre>	update	_bluetooth_salt_verifier()			581	
<pre>update_bluetooth_salt_verifier()     (digi.xbee.devices.CellularDevice</pre>	<pre>update_bluetooth_salt_verifier()     (digi.xbee.devices.CellularDevice</pre>		(digi.xbee.devices.AbstractXBeeDevices)	ce	update	_device_data_from()	
<pre>(digi.xbee.devices.CellularDevice 707</pre>	<pre>(digi.xbee.devices.CellularDevice</pre>		method), 489			(digi.xbee.devices.DigiPointDevice	method),
<pre>1707 update_bluetooth_salt_verifier()     (digi.xbee.devices.DigiMeshDevice</pre>	<pre>707 update_bluetooth_salt_verifier()   (digi.xbee.devices.DigiMeshDevice method), 679 update_bluetooth_salt_verifier()   (digi.xbee.devices.DigiMeshDevice method), 678  update_bluetooth_salt_verifier()   (digi.xbee.devices.DigiPointDevice method), 679  update_bluetooth_salt_verifier()   (digi.xbee.devices.IPDevice method), 679  update_bluetooth_salt_verifier()   (digi.xbee.devices.IPDevice method), 679  update_bluetooth_salt_verifier()   (digi.xbee.devices.Raw802Device method), 679  update_bluetooth_salt_verifier()   (digi.xbee.devices.RemoteDigiMeshDevice method), 851  (digi.xbee.devices.NBIoTDevice method), 851  (digi.xbee.devices.RemoteDigiPointDevice method), 860  (digi.xbee.devices.RemoteDigiPointDevice method), 861  (digi.xbee.devices.RemoteDigiPointDevice method), 861</pre>	update	_bluetooth_salt_verifier()			614	
<pre>update_bluetooth_salt_verifier()     (digi.xbee.devices.DigiMeshDevice</pre>	<pre>update_bluetooth_salt_verifier()     (digi.xbee.devices.DigiMeshDevice</pre>		(digi.xbee.devices.CellularDevice	method),	update		
(digi.xbee.devices.DigiMeshDevicemethod), 581(digi.xbee.devices.LPWANDevicemethod), 734update_bluetooth_salt_verifier() (digi.xbee.devices.DigiPointDeviceupdate_device_data_from() (digi.xbee.devices.NBIoTDevicemethod), 760update_bluetooth_salt_verifier() (digi.xbee.devices.IPDevice method), 679 (digi.xbee.devices.Raw802Deviceupdate_devices.Raw802Devicemethod), 748 (digi.xbee.devices.Raw802Deviceupdate_bluetooth_salt_verifier() 733 update_bluetooth_salt_verifier() (digi.xbee.devices.NBIoTDevicemethod), method), 851 (digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier() (digi.xbee.devices.NBIoTDevicemethod), method), 851 (digi.xbee.devices.RemoteDigiPointDevice	(digi.xbee.devices.DigiMeshDevicemethod),(digi.xbee.devices.LPWANDevicemethod),581734update_bluetooth_salt_verifier()update_device_data_from()method),(digi.xbee.devices.DigiPointDevicemethod),760update_bluetooth_salt_verifier()update_device_data_from()method),(digi.xbee.devices.IPDevice method), 679(digi.xbee.devices.Raw802Devicemethod),(digi.xbee.devices.LPWANDevicemethod),548(digi.xbee.devices.LPWANDevicemethod),update_device_data_from()(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method), 851(digi.xbee.devices.NBIoTDevicemethod),update_device_data_from()(digi.xbee.devices.RemoteDigiPointDevice		,			-	679
T34  update_bluetooth_salt_verifier()     (digi.xbee.devices.DigiPointDevice method),     613  update_bluetooth_salt_verifier()     (digi.xbee.devices.IPDevice method),     (digi.xbee.devices.IPDevice method),     (digi.xbee.devices.Raw802Device method),	581734update_bluetooth_salt_verifier()update_device_data_from()update_devices.NBIoTDevicemethod),613760update_bluetooth_salt_verifier()update_device_data_from()method),(digi.xbee.devices.IPDevice method), 679(digi.xbee.devices.Raw802Devicemethod),update_bluetooth_salt_verifier()548(digi.xbee.devices.Rem802Devicemethod),(733update_device_data_from()(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method), 851(digi.xbee.devices.RemoteDigiPointDevice(digi.xbee.devices.NBIoTDevicemethod),(digi.xbee.devices.RemoteDigiPointDevice	update			update		
(digi.xbee.devices.DigiPointDevicemethod),(digi.xbee.devices.NBIoTDevicemethod),613760update_bluetooth_salt_verifier()update_device_data_from()(digi.xbee.devices.Raw802Devicemethod),update_bluetooth_salt_verifier()548(digi.xbee.devices.LPWANDevicemethod),400update_bluetooth_salt_verifier()update_device_data_from()(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method),851(digi.xbee.devices.NBIoTDevicemethod),update_device_data_from()(digi.xbee.devices.RemoteDigiPointDevice	(digi.xbee.devices.DigiPointDevicemethod),(digi.xbee.devices.NBIoTDevicemethod),613760update_bluetooth_salt_verifier()update_device_data_from()(digi.xbee.devices.Raw802Devicemethod),update_bluetooth_salt_verifier()548(digi.xbee.devices.LPWANDevicemethod),update_device_data_from()733(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method),851(digi.xbee.devices.NBIoTDevicemethod),update_device_data_from()760(digi.xbee.devices.RemoteDigiPointDevice			method),			method),
613  update_bluetooth_salt_verifier() update_device_data_from()  (digi.xbee.devices.IPDevice method), 679  update_bluetooth_salt_verifier() 548  (digi.xbee.devices.LPWANDevice method), 4733  update_bluetooth_salt_verifier() (digi.xbee.devices.RemoteDigiMeshDevice)  update_bluetooth_salt_verifier() method), 851  (digi.xbee.devices.NBIoTDevice method), 460  (digi.xbee.devices.RemoteDigiPointDevice)	613760update_bluetooth_salt_verifier()update_device_data_from()(digi.xbee.devices.IPDevice method), 679(digi.xbee.devices.Raw802Device method),update_bluetooth_salt_verifier()548(digi.xbee.devices.LPWANDevice method),update_device_data_from()733(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method), 851(digi.xbee.devices.NBIoTDevice method)update_device_data_from()760(digi.xbee.devices.RemoteDigiPointDevice	update			update		
(digi.xbee.devices.IPDevice method), 679(digi.xbee.devices.Raw802Device method),update_bluetooth_salt_verifier()548(digi.xbee.devices.LPWANDevice method),update_device_data_from()733(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method), 851(digi.xbee.devices.NBIoTDevice method),update_device_data_from()760(digi.xbee.devices.RemoteDigiPointDevice	<pre>(digi.xbee.devices.IPDevice method), 679</pre>			method),		, <sup>-</sup>	method),
<pre>(digi.xbee.devices.IPDevice method), 679</pre>	<pre>(digi.xbee.devices.IPDevice method), 679</pre>	update	_bluetooth_salt_verifier()		update	_device_data_from()	
(digi.xbee.devices.LPWANDevicemethod),update_device_data_from()733(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method), 851(digi.xbee.devices.NBIoTDevicemethod),update_device_data_from()760(digi.xbee.devices.RemoteDigiPointDevice	(digi.xbee.devices.LPWANDevicemethod),update_device_data_from()733(digi.xbee.devices.RemoteDigiMeshDeviceupdate_bluetooth_salt_verifier()method), 851(digi.xbee.devices.NBIoTDevicemethod),update_device_data_from()760(digi.xbee.devices.RemoteDigiPointDevice					(digi.xbee.devices.Raw802Device	method),
733	733	update					
<pre>update_bluetooth_salt_verifier()</pre>	<pre>update_bluetooth_salt_verifier()</pre>		(digi.xbee.devices.LPWANDevice	method),	update	_device_data_from()	
(digi.xbee.devices.NBIoTDevice method), update_device_data_from() 760 (digi.xbee.devices.RemoteDigiPointDevice	(digi.xbee.devices.NBIoTDevice method), update_device_data_from() 760 (digi.xbee.devices.RemoteDigiPointDevice		733			(digi.xbee.devices.RemoteDigiMeshD	)evice
760 (digi.xbee.devices.RemoteDigiPointDevice	760 (digi.xbee.devices.RemoteDigiPointDevice	update					
				method),	update		
	update_bluetooth_salt_verifier() method), 870					-	evice -
update_bluetooth_salt_verifier() method), 870		update	_bluetooth_salt_verifier()			method), 870	

update_device_data_from() (digi.xbee.devices.RemoteRaw802	2Device	760 update_firmware()
method), 831		(digi.xbee.devices.Raw802Device method),
<pre>update_device_data_from()</pre>		548
(digi.xbee.devices.RemoteXBeeDe	evice	update_firmware()
method), 813		(digi.xbee.devices.RemoteDigiMeshDevice
<pre>update_device_data_from()</pre>		method), 851
(digi.xbee.devices.RemoteZigBeel	Device	<pre>update_firmware()</pre>
method), $890$		(digi.xbee.devices.RemoteDigiPointDevice
<pre>update_device_data_from()</pre>		method), 870
(digi.xbee.devices.WiFiDevice me	thod), 794	<pre>update_firmware()</pre>
<pre>update_device_data_from()</pre>		(digi.x bee. devices. Remote Raw 802 Device
(digi.xbee.devices.XBeeDevice	method),	method), 832
518		<pre>update_firmware()</pre>
<pre>update_device_data_from()</pre>		(digi.xbee.devices.RemoteXBeeDevice
(digi.xbee.devices.ZigBeeDevice	method),	method), 813
652	,,	update_firmware()
<pre>update_filesystem_image()</pre>		(digi.xbee.devices.RemoteZigBeeDevice
(digi.xbee.devices.RemoteDigiMe	shDevice	method), 891
method), 851	SHECVICE	update_firmware() (digi.xbee.devices.WiFiDevice
		method), 794
<pre>update_filesystem_image()</pre>		
(digi.xbee.devices.RemoteDigiPoi	ntDevice	update_firmware() (digi.xbee.devices.XBeeDevice
method), 870		method), 518
<pre>update_filesystem_image()</pre>		<pre>update_firmware()</pre>
(digi.xbee.devices.RemoteRaw802	2Device	(digi.xbee.devices.ZigBeeDevice method),
method), 831		652
<pre>update_filesystem_image()</pre>		<pre>update_firmware()</pre>
(digi.xbee.devices.RemoteXBeeDe	evice	(digi.xbee.serial.XBeeSerialPort method),
method), 797		1033
<pre>update_filesystem_image()</pre>		<pre>update_local_firmware() (in module</pre>
(digi.xbee.devices.RemoteZigBee)	Device	digi.xbee.firmware), 978
method), 890		update_nodes() (digi.xbee.devices.DigiMeshNetwork
update_firmware()		method), 940
	Communication	Interfacee_nodes() (digi.xbee.devices.DigiPointNetwork
method), 472		method), 952
update_firmware()		update_nodes() (digi.xbee.devices.Raw802Network
(digi.xbee.devices.AbstractXBeeD	Device	method), 928
method), 490	evice	
		update_nodes() (digi.xbee.devices.XBeeNetwork method), 895
update_firmware()		
(digi.xbee.devices.CellularDevice	method),	update_nodes() (digi.xbee.devices.ZigBeeNetwork
707		method), 916
<pre>update_firmware()</pre>		<pre>update_remote_filesystem() (in module</pre>
(digi.xbee.devices.DigiMeshDevic	ce method),	digi.xbee.firmware), 979
581		<pre>update_remote_filesystem_image() (in mod-</pre>
<pre>update_firmware()</pre>		ule digi.xbee.filesystem), 975
(digi.xbee.devices.DigiPointDevic	e method),	<pre>update_remote_firmware() (in module</pre>
614		digi.xbee.firmware), 979
update_firmware() (digi.xbee.dev	ices.IPDevice	UpdateConfigurer (class in digi.xbee.firmware),
method), 679		976
update_firmware()		UpdateProfileException, 990
(digi.xbee.devices.LPWANDevice	method),	UpdateProgressStatus (class in
734	,,	digi.xbee.models.status), 232
update_firmware()		USE_BROADCAST_PAN_ID
(digi.xbee.devices.NBIoTDevice	method),	(digi.xbee.models.options.TransmitOptions
(anguino conde rices, 11 Dio 1 De rice	memou),	(argumo como acis.opiions. 1 ransinii opiions

```
attribute), 210
                                                                                                           with_traceback() (digi.xbee.exception.InvalidConfigurationException
USE_CURRENT_OFFSET
                                                                                                                              method), 956
                  (digi.xbee.models.filesystem.ReadFileCmdRequestwith_traceback() (digi.xbee.exception.InvalidOperatingModeExcepti
                 attribute), 150
                                                                                                                              method), 957
USE CURRENT OFFSET
                                                                                                           with_traceback() (digi.xbee.exception.InvalidPacketException
                  (digi.xbee.models.filesystem.WriteFileCmdRequest
                                                                                                                              method), 957
                                                                                                           with traceback() (digi.xbee.exception.OperationNotSupportedExcep
                  attribute), 154
USE EXTENDED TIMEOUT
                                                                                                                              method), 957
                  (digi.xbee.models.options.TransmitOptions
                                                                                                            with_traceback() (digi.xbee.exception.RecoveryException
                  attribute), 211
                                                                                                                              method), 958
user_desc_supported
                                                                                                           with_traceback() (digi.xbee.exception.TimeoutException
                  (digi.xbee.models.zdo.NodeDescriptor
                                                                                                  at-
                                                                                                                              method), 957
                 tribute), 234
                                                                                                           with_traceback() (digi.xbee.exception.TransmitException
UserDataRelayMessage
                                                                       (class
                                                                                                   in
                                                                                                                              method), 957
                  digi.xbee.models.message), 207
                                                                                                            with_traceback() (digi.xbee.exception.XBeeDeviceException
UserDataRelayOutputPacket
                                                                              (class
                                                                                                                              method), 956
                                                                                                   in
                  digi.xbee.packets.relay), 376
                                                                                                           with_traceback() (digi.xbee.exception.XBeeException
UserDataRelayPacket
                                                                                                                             method), 956
                                                                      (class
                                                                                                   in
                  digi.xbee.packets.relay), 373
                                                                                                           with_traceback() (digi.xbee.exception.XBeeSocketException
                                                                                                                              method), 958
                                                                                                           with_traceback() (digi.xbee.fileSystemException
                                                                                                                              method), 959
value (digi.xbee.profile.XBeeProfileSetting attribute),
                                                                                                            with_traceback() (digi.xbee.filesystem.FileSystemNotSupportedExcept
                  989
                                                                                                                              method), 959
version (digi.xbee.profile.XBeeProfile attribute), 991
                                                                                                           with_traceback() (digi.xbee.profile.ReadProfileException
VolFormatCmdRequest
                                                                      (class
                                                                                                                              method), 990
                 digi.xbee.models.filesystem), 188
                                                                                                           with_traceback() (digi.xbee.profile.UpdateProfileException
VolFormatCmdResponse
                                                                       (class
                                                                                                    in
                                                                                                                              method), 990
                  digi.xbee.models.filesystem), 190
                                                                                                            with_traceback() (digi.xbee.util.xmodem.XModemCancelException
VolStatCmdRequest
                                                                    (class
                                                                                                                              method), 468
                  digi.xbee.models.filesystem), 185
                                                                                                           with_traceback() (digi.xbee.util.xmodem.XModemException
VolStatCmdResponse
                                                                     (class
                                                                                                                              method), 468
                  digi.xbee.models.filesystem), 186
                                                                                                           WRITE (digi.xbee.models.options.FileOpenRequestOption
W
                                                                                                                              attribute), 215
\verb|wait_for_frame|()| (\textit{digi.xbee.comm\_interface.XBeeCommunicarion in the face } (\textit{digi.xbee.devices.AbstractXBeeDevice})| (\textit{digi.xbee.devices.AbstractXBeeDevices})| (\textit{digi.xbee.devices.AbstractXBeeDevice})| (\textit{digi.xbee.devices.AbstractXBeeDevices})| (\textit{digi.xbee.devices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices.AbstractXBeeDevices
                                                                                                                              method), 475
                  method), 470
                                                                                                            write_changes() (digi.xbee.devices.CellularDevice
wait_for_frame() (digi.xbee.serial.XBeeSerialPort
                                                                                                                              method), 708
                 method), 1031
                                                                                                           write_changes() (digi.xbee.devices.DigiMeshDevice
wait_until_started()
                                                                                                                             method), 582
                  (digi.xbee.reader.PacketListener
                                                                                       method),
                                                                                                            write_changes() (digi.xbee.devices.DigiPointDevice
                                                                                                                              method), 614
WiFiAssociationIndicationStatus (class in
                                                                                                            write_changes()
                                                                                                                                                                 (digi.xbee.devices.IPDevice
                  digi.xbee.models.status), 226
                                                                                                                              method), 680
WiFiDevice (class in digi.xbee.devices), 761
                                                                                                           write_changes() (digi.xbee.devices.LPWANDevice
WiFiEncryptionType
                                                                                                                              method), 734
                  digi.xbee.models.accesspoint), 130
\label{lem:with_traceback} \textbf{with\_traceback()} \ (\textit{digi.xbee.exception.ATCommandException-} \textbf{changes()}
                                                                                                                                                         (digi.xbee.devices.NBIoTDevice
                                                                                                                             method), 761
                 method), 956
{\tt with\_traceback}~()~(\textit{digi.xbee.exception.CommunicationException}^{\verb|communicationException|}{}^{\verb|communicationException})~(\textit{digi.xbee.devices.Raw802Device})
                                                                                                                              method), 549
                  method), 956
                                                                                                                            changes () (digi.xbee.devices.RemoteDigiMeshDevice
with_traceback() (digi.xbee.exception.ConnectionException-
                                                                                                                              method), 852
                  method), 956
\label{lem:with_traceback} with\_\texttt{traceback} \ () \ (\textit{digi.xbee.exception.FirmwareUpddieException} \ \_\texttt{changes} \ () \ (\textit{digi.xbee.devices.RemoteDigiPointDevice} \ )
                                                                                                                              method), 871
                 method), 958
```

```
write_changes() (digi.xbee.devices.RemoteRaw802Dexir&bit_source_addr
                            method), 832
                                                                                                                                                                                                           (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
write_changes() (digi.xbee.devices.RemoteXBeeDevice
                                                                                                                                                                                                           attribute), 451
                            method), 813
                                                                                                                                                                              x16bit_updater_addr
write_changes() (digi.xbee.devices.RemoteZigBeeDevice
                                                                                                                                                                                                           (digi.xbee.packets.zigbee.OTAFirmwareUpdateStatusPacket
                            method), 891
                                                                                                                                                                                                           attribute), 460
write changes()
                                                                               (digi.xbee.devices.WiFiDevice x64bit dest addr (digi.xbee.packets.common.ExplicitAddressingPack
                            method), 794
                                                                                                                                                                                                           attribute), 300
write_changes()
                                                                             method), 519
                                                                                                                                                                                                           attribute), 274
write_changes()
                                                                       (digi.xbee.devices.ZigBeeDevice x64bit_dest_addr(digi.xbee.packets.common.TransmitPacket
                                                                                                                                                                                                           attribute), 283
                            method), 653
write\_file() (digi.xbee.filesystem.FileSystemManager x 64bit\_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.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.FileSyst
                            method), 962
                                                                                                                                                                                                           attribute), 336
write_frame()(digi.xbee.comm_interface.XBeeCommunicationInterface_addr(digi.xbee.packets.raw.TX64Packet
                             method), 471
                                                                                                                                                                                                           attribute), 350
                                                                        (digi.xbee.serial.XBeeSerialPort x64bit_dest_addr(digi.xbee.packets.zigbee.CreateSourceRoutePacke
write_frame()
                            method), 1031
                                                                                                                                                                                                           attribute), 455
WriteFileCmdRequest
                                                                                                                 (class
                                                                                                                                                                             x64bit_source_addr
                            digi.xbee.models.filesystem), 154
                                                                                                                                                                                                           (digi.xbee.packets.common.ExplicitRXIndicatorPacket
WriteFileCmdResponse
                                                                                                                   (class
                                                                                                                                                                 in
                                                                                                                                                                                                           attribute), 304
                            digi.xbee.models.filesystem), 155
                                                                                                                                                                              x64bit_source_addr
                                                                                                                                                                                                           (digi.xbee.packets.common.IODataSampleRxIndicatorPacket
X
                                                                                                                                                                                                           attribute), 293
x16bit_dest_addr(digi.xbee.packets.common.ExplicitAddressingPackete_addr
                                                                                                                                                                                                           (digi.xbee.packets.common.ReceivePacket
                             attribute), 301
x16bit_dest_addr(digi.xbee.packets.common.RemoteATCommandiplitte), 271
                                                                                                                                                                              x64bit_source_addr
                            attribute), 275
\verb|x16bit_dest_addr| (digi.xbee.packets.common.TransmitPacket| (digi.xbee.packets.common.RemoteATCommandResponsePacket| (digi.xbee.packets.common.RemoteATCommandResponsePackets| (digi.xbee.packets.common.RemoteATCommandResponsePackets| (digi.xbee.packets.common.RemoteATCommandResponsePackets| (digi.xbee.packets.common.RemoteATCommandResponsePackets| (digi.xbee.packets) (digi.xbee.packets| (digi.xbe
                                                                                                                                                                                                           attribute), 279
                             attribute), 283
x16bit_dest_addr(digi.xbee.packets.common.TransmiXfamisPackerurce_addr
                                                                                                                                                                                                           (digi.xbee.packets.filesystem.RemoteFSResponsePacket
                            attribute), 287
                                                                                                                                                                                                           attribute), 341
x16bit_dest_addr(digi.xbee.packets.raw.TX16Packet
                                                                                                                                                                             x64bit_source_addr
                            attribute), 354
at-
                                                                                                                                                                                                           tribute), 367
                            attribute), 455
                                                                                                                                                                              x64bit source addr
x16bit_source_addr
                                                                                                                                                                                                           (digi.xbee.packets.raw.RX64Packet attribute),
                            (digi.xbee.packets.common.ExplicitRXIndicatorPacket
                                                                                                                                                                                                           360
                            attribute), 304
                                                                                                                                                                             x64bit_source_addr
x16bit_source_addr
                             (digi.xbee.packets.common. IOD at a Sample Rx Indicator Packet digi.xbee.packets.zigbee. OTA Firmware Update Status Packet digi.xbee.packet digi.xbee.pa
                                                                                                                                                                                                           attribute), 460
                            attribute), 294
                                                                                                                                                                              x64bit source addr
x16bit_source_addr
                                                                                                                                                                                                           (digi.xbee.packets.zigbee.RouteRecordIndicatorPacket
                            (digi.xbee.packets.common.ReceivePacket
                                                                                                                                                                                                           attribute), 451
                            attribute), 271
                                                                                                                                                                              x64bit_target_addr
x16bit_source_addr
                             (digi.xbee.packets.common. Remote ATC ommand Response Packets. zigbee. OTA Firmware Update Status Packets and the property of the property o
                                                                                                                                                                                                           attribute), 461
                            attribute), 279
                                                                                                                                                                             xbee
                                                                                                                                                                                                           (digi.xbee.filesystem.FileSystemManager
x16bit_source_addr
                                                                                                                                                                                                           tribute), 960
                            (digi.xbee.packets.raw.RX16IOPacket
                                                                                                                                                              at-
                                                                                                                                                                             xbee (digi.xbee.firmware.FwUpdateTask attribute), 978
                            tribute), 372
                                                                                                                                                                             xbee (digi.xbee.profile.ProfileUpdateTask attribute), 994
x16bit source addr
                                                                                                                                                                             xbee (digi.xbee.sender.SyncRequestSender attribute),
                             (digi.xbee.packets.raw.RX16Packet attribute),
                                                                                                                                                                                                           1029
                             364
```

```
XBee16BitAddress
                                (class
                                                in
        digi.xbee.models.address), 200
XBee64BitAddress
                                                in
        digi.xbee.models.address), 202
XBeeAPIPacket (class in digi.xbee.packets.base), 246
XBeeCommunicationInterface
        digi.xbee.comm interface), 470
XBeeDevice (class in digi.xbee.devices), 491
XBeeDeviceException, 956
XBeeEvent (class in digi.xbee.reader), 995
XBeeException, 956
XBeeIMEIAddress
                               (class
                                                in
        digi.xbee.models.address), 203
XBeeLocalInterface
                                 (class
                                                in
        digi.xbee.models.options), 213
XBeeMessage (class in digi.xbee.models.message), 204
XBeeNetwork (class in digi.xbee.devices), 892
XBeePacket (class in digi.xbee.packets.base), 244
XBeeProfile (class in digi.xbee.profile), 990
XBeeProfileSetting (class in digi.xbee.profile),
XBeeProtocol (class in digi.xbee.models.protocol),
        216
XBeeQueue (class in digi.xbee.reader), 1025
XBeeSerialPort (class in digi.xbee.serial), 1030
XBeeSettingFormat (class in digi.xbee.profile), 988
XBeeSettingType (class in digi.xbee.profile), 987
XBeeSocketException, 957
xml_path
             (digi.xbee.firmware.FwUpdateTask
                                               at-
        tribute), 978
XModemCancelException, 468
XModemException, 468
Ζ
ZigBeeDevice (class in digi.xbee.devices), 615
ZigBeeNetwork (class in digi.xbee.devices), 904
ZigbeeRegisterStatus
                                   (class
                                                in
        digi.xbee.models.status), 228
```