



Digi XBee3[®] Cellular LTE-M

Smart Modem

User Guide

Revision history—900002258

Revision	Date	Description
A	December 2017	Initial release of the document.

Trademarks and copyright

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

© 2017 Digi International Inc. All rights reserved.

Disclaimers

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International. Digi provides this document “as is,” without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

Warranty

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

www.digi.com/howtobuy/terms

Send comments

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

Customer support

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

Contents

Digi XBee3 Cellular LTE-M Global Smart Modem User Guide

Applicable firmware and hardware	9
SIM cards	9

Getting started with the XBee Smart Modem Development Kit

Identify the kit contents	11
XBIB-U-DEV reference	12
Cellular service	13
Connect the hardware	14
Configure the device using XCTU	15
Add a device	15
Check for cellular registration and connection	16
Update to the latest firmware	16
Send an SMS message to a phone	18
Debugging	19
Connect to the ELIZA server	20
Debugging	21
Connect to the echo server	22
Debugging	23
Connect to the Daytime server	24
Debugging	25
Connect to a TCP/IP address	26
Debugging	27
Perform a (GET) HTTP request	28
Debugging	29
Get started with MQTT	30
Example: MQTT connect	30
Send a connect packet	32
Example: send messages (publish) with MQTT	34
Example: receive messages (subscribe) with MQTT	34
Use MQTT over the XBee Cellular Modem with a PC	35
Get started with CoAP	39
CoAP terms	39
CoAP quick start example	39
Configure the device	40
Example: manually perform a CoAP request	40
Example: use Python to generate a CoAP message	41
Configure the XBee Smart Modem using Digi Remote Manager	44
Create a Remote Manager account	44

Get the XBee Smart Modem IMEI number	44
Add a XBee Smart Modem to Remote Manager	44
Update the firmware	45
Software libraries	45

Get started with MicroPython

About MicroPython	47
Why use MicroPython	47
MicroPython on the XBee Smart Modem	47
Use XCTU to enter the MicroPython environment	47
Use the MicroPython Terminal in XCTU	48
Example: hello world	48
Example: turn on an LED	48
Example: code a request help button	49
Enter MicroPython paste mode	50
Catch a button press	50
Send a text (SMS) when the button is pressed	52
Add the time the button was pressed	53
Exit MicroPython mode	54
Other terminal programs	54
Tera Term for Windows	54
Use picocom in Linux	55

Technical specifications

Interface and hardware specifications	58
Cellular RF characteristics	58
Bluetooth RF characteristics	58
Cellular Networking specifications	58
Power requirements	59
Power consumption	59
Electrical specifications	59
Regulatory approvals	60

Hardware

Mechanical drawings	62
Pin signals	62
Pin connection recommendations	64
RSSI PWM	64
SIM card	64
The Associate LED	65

Antenna recommendations

Antenna placement	67
-------------------------	----

Design recommendations

Power supply considerations	69
Minimum connection diagram	69

Heat considerations and testing	69
Heat sink guidelines	71
Bolt-down style	71
Adhesive style heat sink	71
Add a fan to provide active cooling	72

Cellular connection process

Connecting	74
Cellular network	74
Data network connection	74
Data communication with remote servers (TCP/UDP)	74
Disconnecting	74

Modes

Select an operating mode	77
Transparent operating mode	78
API operating mode	78
Bypass operating mode	78
Enter Bypass operating mode	78
Leave Bypass operating mode	79
Restore cellular settings to default in Bypass operating mode	79
USB direct mode	79
Configure the data pins	79
Enable USB direct mode	79
Enable the VBUS option	80
Command mode	80
Enter Command mode	80
Send AT commands	81
Apply command changes	81
Make command changes permanent	81
Exit Command mode	82

Sleep modes

About sleep modes	84
Normal mode	84
Pin sleep mode	84
Cyclic sleep mode	84
Cyclic sleep with pin wake up mode	84
Airplane mode	84
SPI mode and sleep pin functionality	84
The sleep timer	85
MicroPython sleep behavior	85

Serial communication

Serial interface	88
Serial data	88
UART data flow	88
Serial buffers	89
CTS flow control	89

RTS flow control	89
Enable UART or SPI ports	89

SPI operation

SPI communications	92
Full duplex operation	93
Low power operation	94
Select the SPI port	94
Force UART operation	95
Data format	95

AT commands

Special commands	97
AC (Apply Changes)	97
FR (Force Reset)	97
RE command	97
WR command	97
Cellular commands	98
PH (Phone Number)	98
S# (ICCID)	98
IM (IMEI)	98
MN (Operator)	98
MV (Modem Firmware Version)	99
DB (Cellular Signal Strength)	99
AN (Access Point Name)	99
AM (Airplane Mode)	99
Network commands	100
IP (IP Protocol)	100
TL (SSL/TLS Protocol Version)	100
TM (IP Client Connection Timeout)	100
TS (IP Server Connection Timeout)	101
DO (Device Options)	101
EQ (Device Cloud FQDN)	102
Addressing commands	102
SH (Serial Number High)	102
SL (Serial Number Low)	102
MY (Module IP Address)	102
P# (Destination Phone Number)	103
N1 (DNS Address)	103
N2 (DNS Address)	103
DL (Destination Address)	103
OD (Operating Destination Address)	104
DE (Destination Port)	104
C0 (Source Port)	104
LA (Lookup IP Address of FQDN)	105
Serial interfacing commands	105
BD (Baud Rate)	105
NB (Parity)	106
SB (Stop Bits)	106
RO (Packetization Timeout)	106
TD (Text Delimiter)	107
FT (Flow Control Threshold)	107

AP (API Enable)	107
I/O settings commands	107
D0 (DIO0/AD0)	108
D1 (DIO1/AD1)	108
D2 (DIO2/AD2)	108
D3 (DIO3/AD3)	109
D4 (DIO4)	109
D5 (DIO5/ASSOCIATED_INDICATOR)	110
D6 (DIO6/RTS)	110
D7 (DIO7/CTS)	110
D8 (DIO8/SLEEP_REQUEST)	111
P0 (DIO10/PWM0 Configuration)	112
P1 (DIO11/PWM1 Configuration)	112
P2 (DIO12 Configuration)	113
P3 (DIO13/DOOUT)	113
P4 (DIO14/DIN)	113
PD (Pull Direction)	114
PR (Pull-up/down Resistor Enable)	114
M0 (PWM0 Duty Cycle)	115
I/O sampling commands	115
TP (Temperature)	115
Sleep commands	116
SM (Sleep Mode)	116
SP (Sleep Period)	116
ST (Wake Time)	116
Command mode options	117
CC (Command Sequence Character)	117
CT (Command Mode Timeout)	117
GT (Guard Times)	117
MicroPython commands	117
PS (Python Startup)	118
PY (MicroPython Command)	118
Firmware version/information commands	119
VR (Firmware Version)	119
VL (Verbose Firmware Version)	119
HV (Hardware Version)	119
AI (Association Indication)	119
DI (Device Cloud Indicator)	120
CI (Protocol/Connection Indication)	120
HS (Hardware Series)	122
CK (Configuration CRC)	122
Execution commands	122
!R (Modem Reset)	122
IS (Force Sample)	123

Operate in API mode

API mode overview	125
Use the AP command to set the operation mode	125
API frame format	125
API operation (AP parameter = 1)	125
API operation with escaped characters (AP parameter = 2)	126
Frame descriptions	129
AT Command - 0x08	129
AT Command: Queue Parameter Value - 0x09	130

Transmit (TX) Request: IPv4 - 0x20	131
AT Command Response - 0x88	132
Transmit (TX) Status - 0x89	133
Modem Status - 0x8A	134
Receive (RX) Packet: IPv4 - 0xB0	135

Socket behavior

Supported sockets	137
Socket timeouts	137
Socket limits in API mode	137
Enable incoming TCP sockets in API mode	137
API mode behavior for outgoing TCP and SSL connections	137
API mode behavior for outgoing UDP data	138
API mode behavior for incoming TCP connections	138
API mode behavior for incoming UDP data	139
Transparent mode behavior for outgoing TCP and SSL connections	139
Transparent mode behavior for outgoing UDP data	139
Transparent mode behavior for incoming TCP connections	140
Transparent mode behavior for incoming UDP connections	140

Troubleshooting

Cannot find the serial port for the device	142
Condition	142
Solution	142
Correct a macOS Java error	144
Condition	144
Solution	144
Unresponsive cellular component in Bypass mode	145
Condition	145
Solution	145
Syntax error at line 1	145
Solution	145
Network connection issues	145
Condition	145
Solution	145
Set the APN value	145

Regulatory information

United States (FCC)	148
OEM labeling requirements	148
FCC notices	148
FCC-approved antennas	149
RF exposure	150
IC (Industry Canada)	150
Labeling requirements	150
RF Exposure	150

Digi XBee3 Cellular LTE-M Global Smart Modem User Guide

The XBee Smart Modem provides OEMs with a simple way to integrate low-power cellular connectivity into their devices. Features include:

- FCC certified and carrier end-device certified
- Excellent coverage and building penetration
- Digi XBee Transparent and API modes simplify design
- Low power consumption optimized for long battery life
- Reduced hardware complexity with only 1 antenna required
- Integrated MicroPython programmability enables custom scripting directly on the modem
- Enhanced with Digi TrustFence™ security framework
- Manage and configure with XCTU and Digi Remote Manager®
- Available with Digi provided SIM cards and data plans

Applicable firmware and hardware

This manual supports the following firmware:

- 311xx

It supports the following hardware:

- XB3-C-A2-UT-xxx

SIM cards

The XBee Smart Modem requires a 4FF (Nano) size SIM card. The SIM interface supports both 1.8 V and 3 V SIM types.

Getting started with the XBee Smart Modem Development Kit

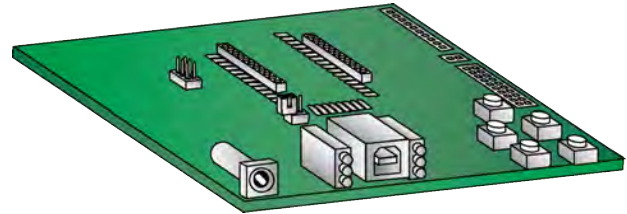
This section describes how to connect the hardware in the XBee Smart Modem Development Kit, and provides some examples you can use to communicate with the device.

Identify the kit contents	11
XBIB-U-DEV reference	12
Cellular service	13
Connect the hardware	14
Configure the device using XCTU	15
Send an SMS message to a phone	18
Connect to the ELIZA server	20
Connect to the echo server	22
Connect to the Daytime server	24
Connect to a TCP/IP address	26
Perform a (GET) HTTP request	28
Get started with MQTT	30
Get started with CoAP	39
Configure the XBee Smart Modem using Digi Remote Manager	44
Software libraries	45

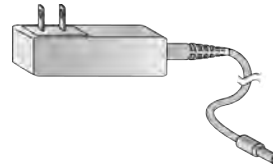
Identify the kit contents

The Developer's kit includes the following:

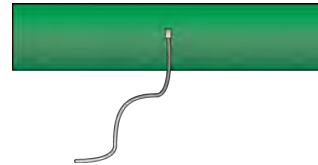
One XBIB-U-DEV board



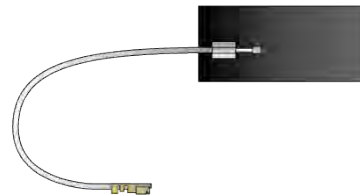
One 12 V power supply



One cellular antenna with U.FL connector



One Bluetooth Low Energy (BLE) antenna
(BLE support is forthcoming but not currently available)



One USB cable



One XBee Smart Modem

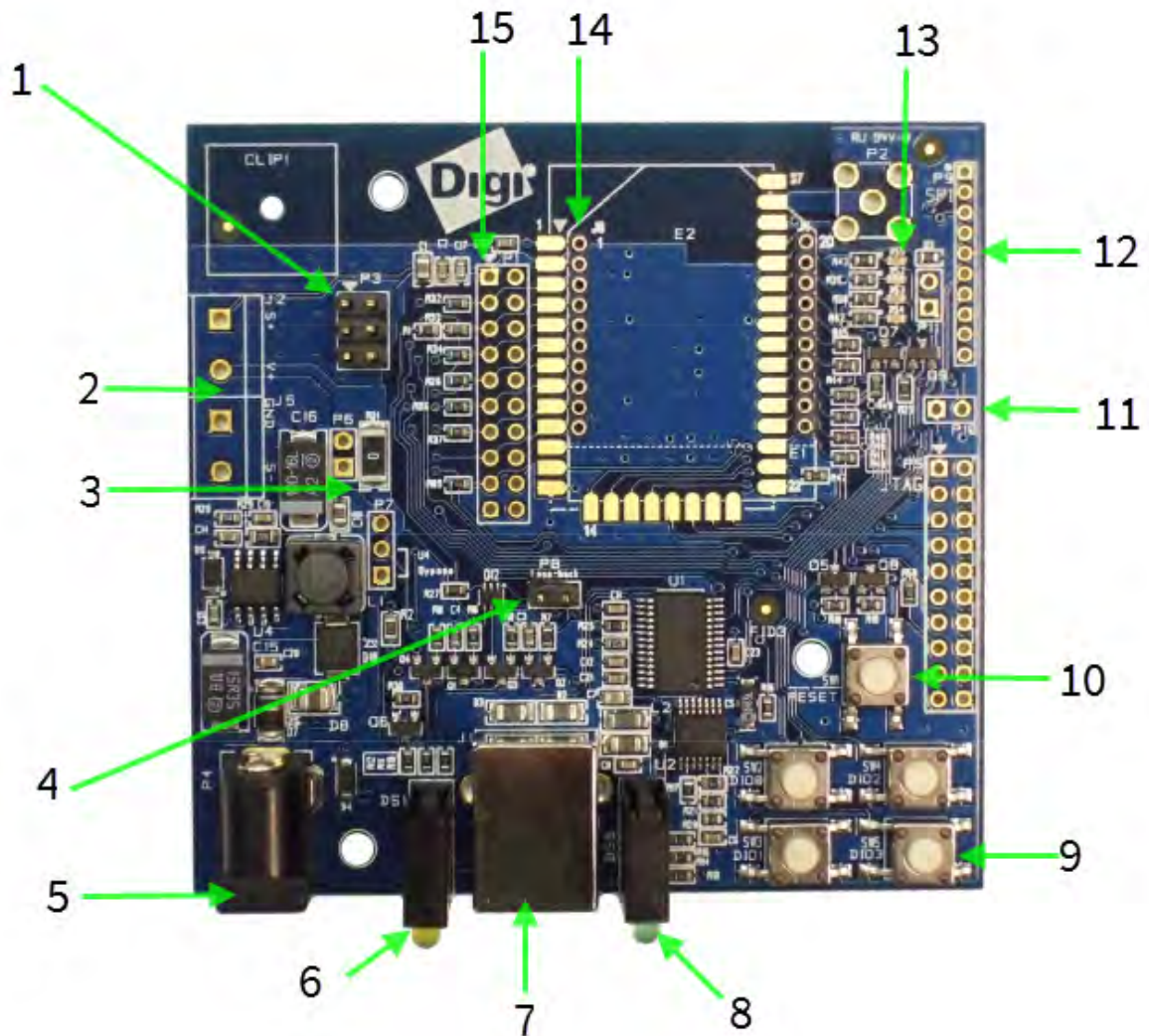


One SIM card



XBIB-U-DEV reference

This picture shows the XBee USB development board and the table that follows explains the callouts in the picture.



Number	Item	Description
1	Programming header	Header used to program XBee Programmable devices.

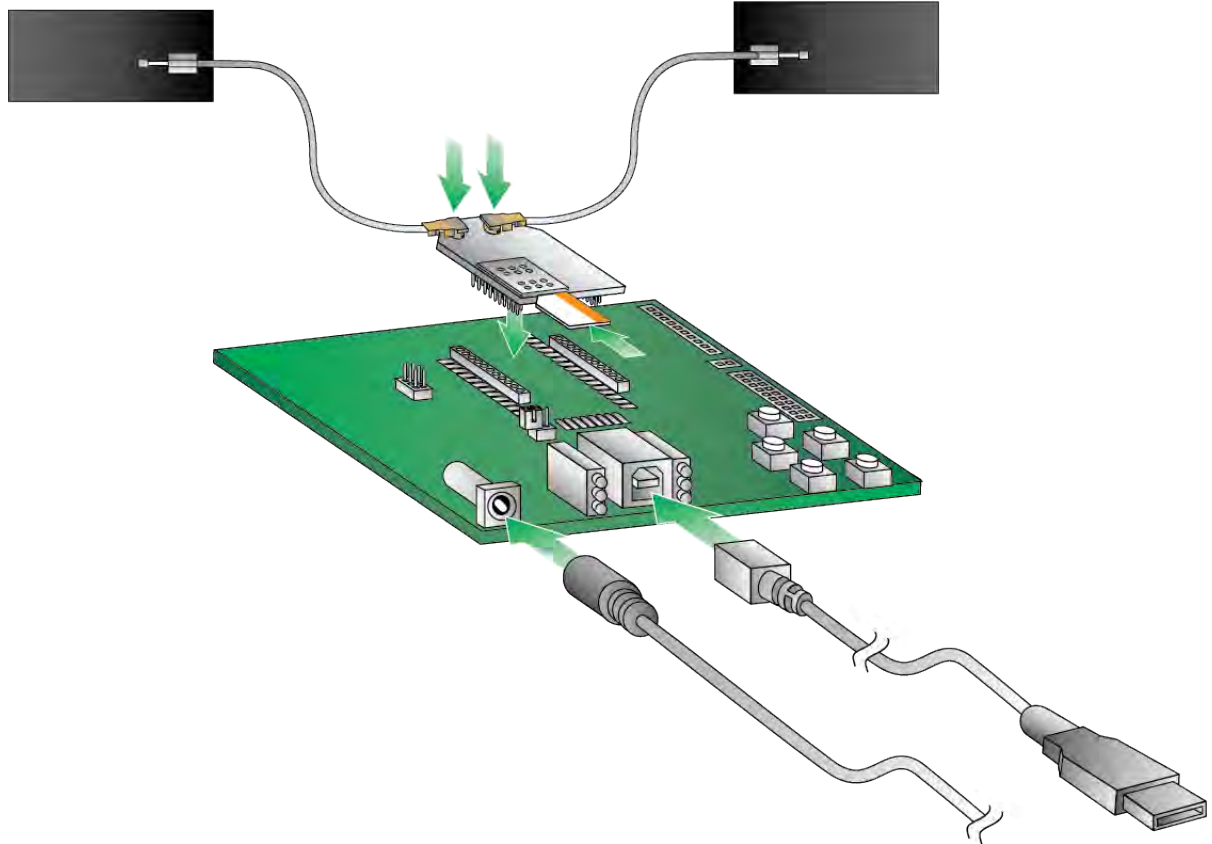
Number	Item	Description
2	Self power module	Advanced users only—voids the warranty. Depopulate R31 to power the device using V+ and GND from J2 and J5. You can connect sense lines to S+ and S- for sensing power supplies.  CAUTION: Voltage is not regulated. Applying the incorrect voltage can cause fire and serious injury. ¹
3	Current testing	Depopulating R31 allows a current probe to be inserted across P6 terminals. The current through P6/R31 powers the device only. Other supporting circuitry is powered by a different trace.
4	Loopback jumper	Populating P8 with a loopback jumper causes serial transmissions both from the device and from the USB to loopback.
5	DC barrel plug: 6-20 V	Greater than 500 mA loads require a DC supply for correct operation. Plug in the external power supply prior to the USB connector to ensure that proper USB communications are not interrupted.
6	LED indicator	Yellow: Modem sending serial/UART data to host. Green: Modem receiving serial/UART data from host. Red: Associate.
7	USB	
8	RSSI indicator	
9	User buttons	Connected to DIO lines for user implementation.
10	Reset button	
11	SPI power	Connect to the power board from 3.3 V.
12	SPI	Only used for surface-mount devices.
13	Indicator LEDs	DS5: ON/ <u>SLEEP</u> DS2: DIO12, the LED illuminates when driven low. DS3: DIO11, the LED illuminates when driven low. DS4: DIO4, the LED illuminates when driven low.
14	Through-hole XBee sockets	
15	20-pin header	Maps to standard through-hole XBee pins.

Cellular service

The XBee Cellular kit includes six months of free cellular service.

¹Powering the board with J2 and J5 without R31 removed can cause shorts if the USB or barrel plug power are connected. Applying too high a voltage destroys electronic circuitry in the device and other board components and/or can cause injury.

Connect the hardware



1. The XBee Smart Modem should already be plugged into the XBIB-U-DEV board.
2. The SIM card should already be inserted into the XBee Smart Modem. If not, install the SIM card into the XBee Smart Modem.



WARNING! Never insert or remove the SIM card while the device is powered!

3. Connect the antennas to the XBee Smart Modem by aligning the U.FL connectors carefully, then firmly pressing straight down to seat the connector. You should hear a snap when the antenna attaches correctly. U.FL is fragile and is not designed for multiple insertions, so exercise caution when connecting or removing the antennas. We recommend using a U.FL removal tool.
4. Plug the 12 V power supply to the power jack on the development board.
5. Connect the USB cable from a PC to the USB port on the development board. The computer searches for a driver, which can take a few minutes to install.

Configure the device using XCTU

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

XCTU does not work directly over an SPI interface.

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

Note If you are on a macOS computer and encounter problems installing XCTU, see [Correct a macOS Java error](#).

Add a device

These instructions show you how to add the XBee Smart Modem to XCTU. If XCTU does not find your serial port, see [Cannot find the serial port for the device](#).

1. Launch XCTU .
2. Click the **Discover radio modules** button .
3. In the **Discover radio devices** dialog, select the serial ports where you want to look for XBee modules, and click **Next**.
4. In the **Set port parameters** window, maintain the default values and click **Finish**.
5. As XCTU locates radio modules, they appear in the **Discovering radio modules** dialog box.

If your module could not be found, XCTU displays the **Could not find any radio module** dialog providing possible reasons why the module could not be added.

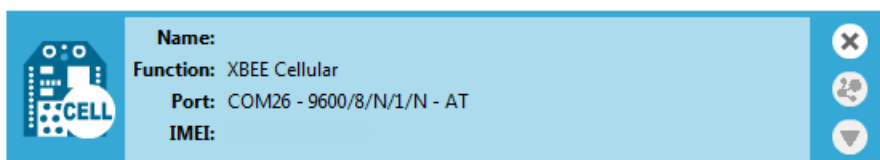
Check for cellular registration and connection

In the following examples, proper cellular network registration and address assignment must occur successfully. The LED on the development board blinks when the XBee Smart Modem is registered to the cellular network; see [The Associate LED](#). If the LED remains solid, registration has not occurred properly. Registration can take several minutes.

Note Make sure you are in an area with adequate cellular network reception or the XBee Smart Modem will not make the connection.

In addition to the LED confirmation, you can check the AT commands below in XCTU to check the registration and connection. To view these commands:

1. Open XCTU and [Add a device](#).
2. Click the **Configuration working mode**  button.
3. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.



4. On the Configuration toolbar, click the **Default** button  to load the default values established by the firmware, and click **Yes** to confirm.

The relevant commands are:

- **AI** (Association Indication) reads zero when the device successfully registers to the cellular network. If it reads 0x23 it is connecting to the Internet; 0x22 means it is registering to the cellular network.
- **MY** (Module IP Address) should display a valid IP address. If it reads **0.0.0.0**, it has not registered yet.

Note To search for an AT command in XCTU, use [the search box](#).

Update to the latest firmware

Firmware is the program code stored in the device's persistent memory that provides the control program for the device. Use XCTU to update the firmware.

1. Click the **Configuration working modes** button .
2. Select a local XBee module from the **Radio Modules** list.
3. Click the **Update firmware** button .

The **Update firmware** dialog displays the available and compatible firmware for the selected XBee module.

4. Select the product family of the XBee module, the function set, and the latest firmware version.

5. Click **Update**. A dialog displays update progress. Click **Show details** for details of the firmware update process.

See [How to update the firmware of your modules](#) in the XCTU User Guide for more information.

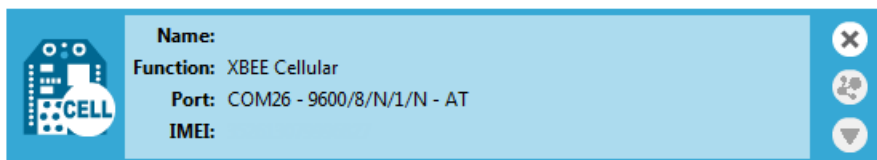
Send an SMS message to a phone

The XBee Smart Modem can send and receive Short Message Service (SMS) transmissions (text messages) while in Transparent mode. This allows you to send and receive text messages to and from an SMS capable device such as a mobile phone.

The following table explains the AT commands that you use in this example.

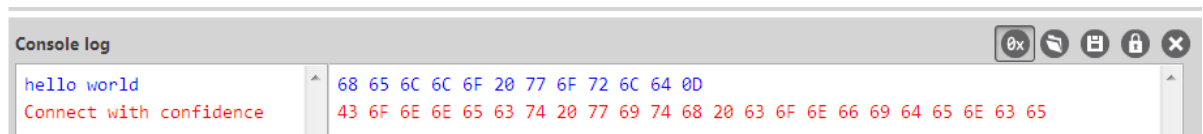
Command	Value	Description
AP (API Enable)	0	Set the device's API mode to Transparent mode.
IP (IP Protocol)	2	Set the expected transmission mode to SMS communication.
P# (Destination Phone Number)	<Target phone number>	The target phone number that you send to, for example, your cellular phone. See P# (Destination Phone Number) for instructions on using this command.
TD (Text Delimiter)	D (0x0D)	The text delimiter to be used for Transparent mode, as an ASCII hex code. No information is sent until this character is entered, unless the maximum number of characters has been reached. Set to zero to disable text delimiter checking. Set to D for a carriage return.
PH (Module's SIM phone number)	Read only	The value that represents your device's phone number as supplied by the SIM card. This is used to send text messages to the device from another cellular device.

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and [Add a device](#).
3. Click the **Configuration working mode**  button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.



5. To switch to SMS communication, in the **IP** field, select **2** and click the **Write** button .
6. To enter your cell phone number, in the **P#** field, type the <**target phone number**> and click the **Write** button. Type the phone number using only numbers, with no dashes. You can use the + prefix if necessary. The target phone number is the phone number you wish to send a text to.
7. In the **TD** field, type **D** and click the **Write** button.
8. Note the number in the **PH** field; it is the XBee Smart Modem phone number, which you see when it sends an SMS to your phone.

9. Click the **Consoles working mode** button  on the toolbar to open a serial console to the device. For instructions on using the Console, see the [AT console](#) topic in the *XCTU User Guide*.
10. Click the **Open** button  to open a serial connection to the device.
11. Click in the left pane of the **Console log**, type **hello world** and press **Enter**. The XBee Smart Modem sends the message to the destination phone number set by the **P#** command.
12. When the phone receives the text, you can see that the sender's phone number matches the value reported by the XBee Smart Modem with the **PH** command.
13. On the phone, reply with the text **connect with confidence** and the XBee Smart Modem outputs this reply from the UART.



Debugging

If you experience problems with the settings in this example, you can load the default settings in XCTU:

1. On the Configuration toolbar, click the **Default** button  to load the default values established by the firmware, and click **Yes** to confirm.
2. Factory settings are loaded but not written to the device. To write them, click the **Write** button  on the toolbar.

Connect to the ELIZA server

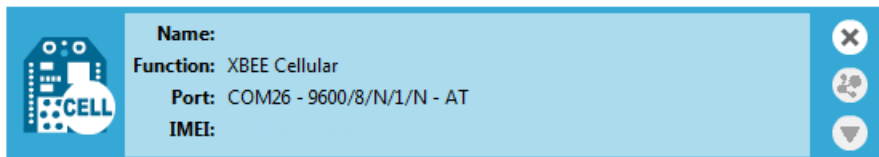
You can use the XBee Smart Modem to chat with the ELIZA Therapist Bot. ELIZA is an artificial intelligence (AI) bot that emulates a therapist and can perform simple conversations.

The following table explains the AT commands that you use in this example.

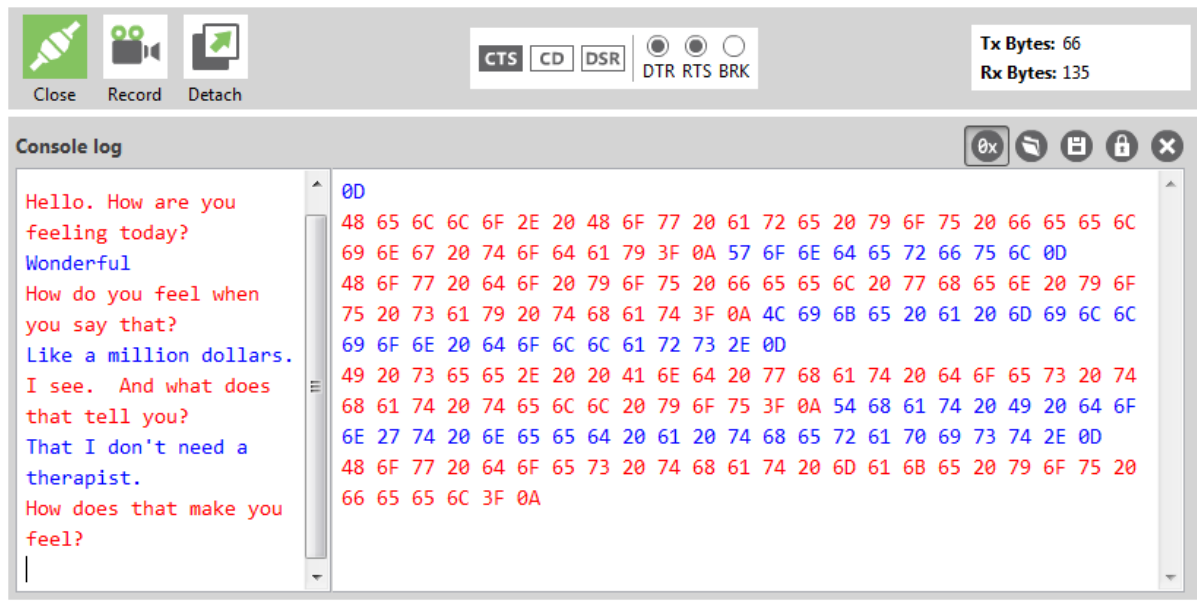
At command	Value	Description
IP (IP Protocol)	1	Set the expected transmission mode to TCP communications.
DL (Destination Address)	52.43.121.77	The target IP address of the Eliza server.
DE (Destination Port)	0x2328	The target port number of the Eliza server.

To communicate with the ELIZA Therapist Bot:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and [Add a device](#).
3. Click the **Configuration working mode**  button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.



5. To switch to TCP communication, in the **IP** field, select 1 and click the **Write** button .
6. To enter the destination address of the ELIZA Therapist Bot, in the **DL** field, type **52.43.121.77** and click the **Write** button.
7. To enter the destination IP port number, in the **DE** field, type **2328** and click the **Write** button.
8. Click the **Consoles working mode**  button on the toolbar to open a serial console to the device. For instructions on using the Console, see the [AT console](#) topic in the *XCTU User Guide*.
9. Click the **Open** button  to open a serial connection to the device.
10. Click in the left pane of the **Console log**, then type in the Console to talk to the ELIZA Therapist Bot. The following screenshot provides an example of this chat.



Debugging

If you experience problems with the settings in this example, you can load the default settings in XCTU:

1. On the Configuration toolbar, click the **Default** button  to load the default values established by the firmware, and click **Yes** to confirm.
2. Factory settings are loaded but not written to the device. To write them, click the **Write** button  on the toolbar.

Connect to the echo server

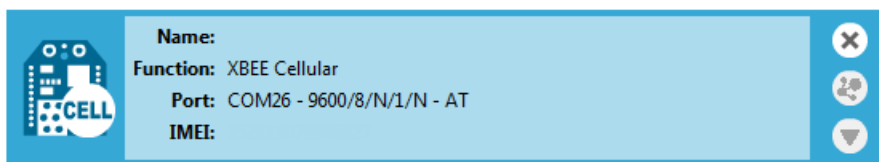
This server echoes back the messages you type.

The following table explains the AT commands that you use in this example.

At command	Value	Description
IP (IP Protocol)	1	Set the expected transmission mode to TCP communications.
TD (Text Delimiter)	D (0x0D)	The text delimiter to be used for Transparent mode, as an ASCII hex code. No information is sent until this character is entered, unless the maximum number of characters has been reached. Set to zero to disable text delimiter checking. Set to D for a carriage return.
DL (Destination Address)	52.43.121.77	The target IP address of the echo server.
DE (Destination Port)	0x2329	The target port number of the echo server.

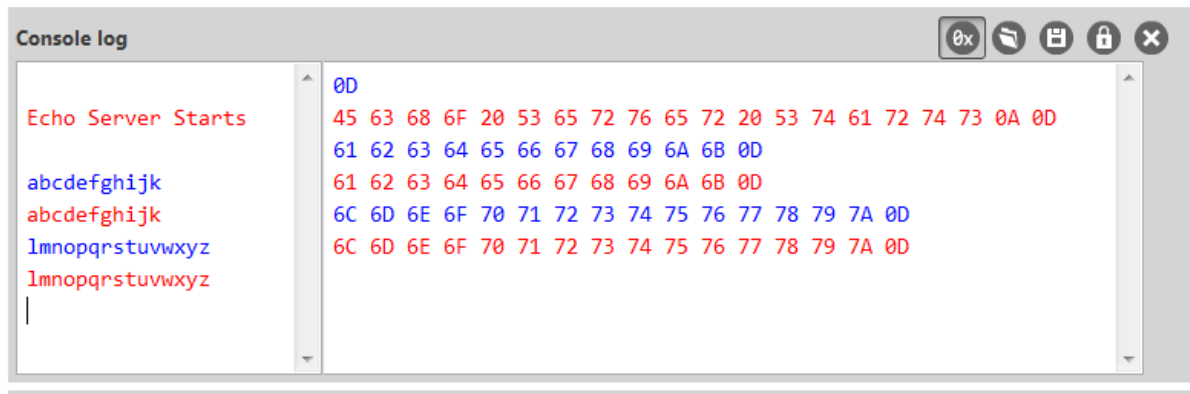
To communicate with the echo server:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and [Add a device](#).
3. Click the **Configuration working mode**  button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.



5. To switch to TCP communication, in the **IP** field, select 1 and click the **Write** button .
6. To enable the XBee Smart Modem to recognize carriage return as a message delimiter, in the **TD** field, type **D** and click the **Write** button.
7. To enter the destination address of the echo server, in the **DL** field, type **52.43.121.77** and click the **Write** button.
8. To enter the destination IP port number, in the **DE** field, type **2329** and click the **Write** button.

9. Click the **Consoles working mode** button  on the toolbar to open a serial console to the device. For instructions on using the Console, see the [AT console](#) topic in the *XCTU User Guide*.
10. Click the **Open** button  to open a serial connection to the device.
11. Click in the left pane of the **Console log**, then type in the Console to talk to the echo server. The following screenshot provides an example of this chat.



Debugging

If you experience problems with the settings in this example, you can load the default settings in XCTU:

1. On the Configuration toolbar, click the **Default** button  to load the default values established by the firmware, and click **Yes** to confirm.
2. Factory settings are loaded but not written to the device. To write them, click the **Write** button  on the toolbar.

Connect to the Daytime server

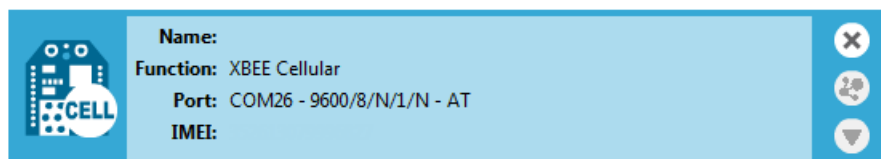
The Daytime server reports the current Coordinated Universal Time (UTC) value responding to any user input.

The following table explains the AT commands that you use in this example.

At command	Value	Description
IP (IP Protocol)	1	Set the expected transmission mode to TCP communications.
DL (Destination Address)	52.43.121.77	The target IP of the Daytime server.
DE (Destination Port)	0x232A	The target port number of the Daytime server.
TD (Text Delimiter)	0	The text delimiter to be used for Transparent mode, as an ASCII hex code. No information is sent until this character is entered, unless the maximum number of characters has been reached. Set to zero to disable text delimiter checking.

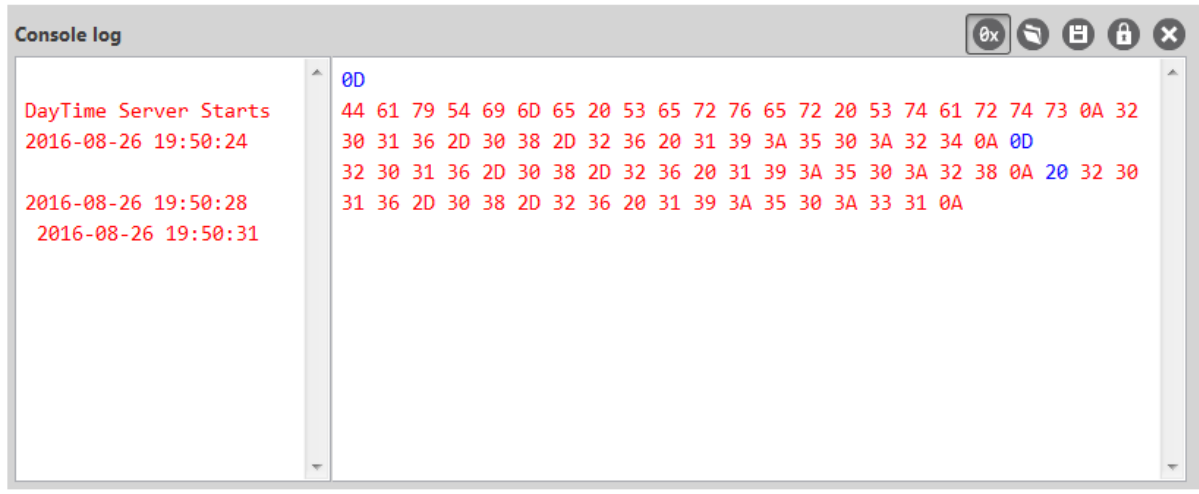
To communicate with the Daytime server:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and [Add a device](#).
3. Click the **Configuration working mode**  button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.



5. To switch to TCP communication, in the **IP** field, select 1 and click the **Write** button .
6. To enter the destination address of the daytime server, in the **DL** field, type **52.43.121.77** and click the **Write** button.
7. To enter the destination IP port number, in the **DE** field, type **232A** and click the **Write** button.
8. To disable text delimiter checking, in the **TD** field, type **0** and click the **Write** button.
9. Click the **Consoles working mode**  button on the toolbar to open a serial console to the device. For instructions on using the Console, see the [AT console](#) topic in the *XCTU User Guide*.

10. Click the **Open** button  to open a serial connection to the device.
11. Click in the left pane of the **Console log**, then type in the Console to query the Daytime server.
The following screenshot provides an example of this chat.



Debugging

If you experience problems with the settings in this example, you can load the default settings in XCTU:

1. On the Configuration toolbar, click the **Default** button  to load the default values established by the firmware, and click **Yes** to confirm.
2. Factory settings are loaded but not written to the device. To write them, click the **Write** button  on the toolbar.

Connect to a TCP/IP address

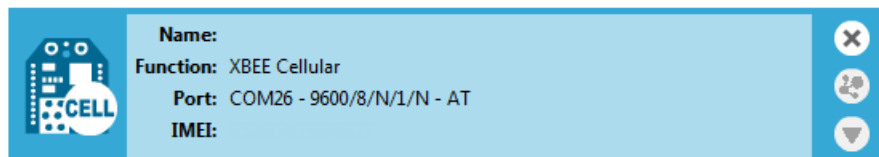
The XBee Smart Modem can send and receive TCP messages while in Transparent mode; see [Transparent operating mode](#).

You can use this example as a template for sending and receiving data from a user. The following table explains the AT commands that you use in this example.

Command	Value	Description
IP (IP Protocol)	1	Set the expected transmission mode to TCP communication.
DL (Destination IP Address)	<Target IP address>	The target IP address that you send and receive from. For example, a data logging server's IP address that you want to send measurements to.
DE (Destination Port)	<Target port number>	The target port number that the device sends the transmission to. This is represented as a hexadecimal value.

To connect to a TCP/IP address:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and [Add a device](#).
3. Click the **Configuration working mode**  button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.



5. In the **IP** field, select 1 and click the **Write** button .
6. In the **DL** field, type the <target IP address> and click the **Write** button. The target IP address is the IP address that you send and receive from.
7. In the **DE** field, type the <target port number>, converted to hexadecimal, and click the **Write** button.
8. Exit Command mode; see [Exit Command mode](#).

After exiting Command mode, any UART data sent to the device is sent to the destination IP address and port number after the **RO (Packetization Timeout)** occurs.

Debugging

If you experience problems with the settings in this example, you can load the default settings in XCTU:

1. On the Configuration toolbar, click the **Default** button  to load the default values established by the firmware, and click **Yes** to confirm.
2. Factory settings are loaded but not written to the device. To write them, click the **Write** button  on the toolbar.

Perform a (GET) HTTP request

You can use the XBee Smart Modem to perform a GET Hypertext Transfer Protocol (HTTP) request using XCTU. This example uses <http://httpbin.org/> as the target website that responds to the HTTP request.

To perform a GET request:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and [Add a device](#).
3. Click the **Configuration working mode**  button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. To enter the destination address of the target website, in the **DL** field, type **httpbin.org** and click the **Write** button .
6. To enter the HTTP request port number, in the **DE** field, type **50** and click the **Write** button. Hexadecimal **50** is 80 in decimal.
7. To switch to TCP communication, in the **IP** field, select **1** and click the **Write** button.
8. To move into Transparent mode, in the **AP** field, select **0** and click the **Write** button.
9. Wait for the **AI** (Association Indication) value to change to **0** (Connected to the Internet).
10. Click the **Consoles working mode** button  on the toolbar.
11. From the AT console, click the **Add new packet button**  in the Send packets dialog. The **Add new packet** dialog appears.
12. Enter the name of the data packet.
13. Type the following data in the **ASCII** input tab:
GET /ip HTTP/1.1
Host: httpbin.org
14. Click the **HEX** input tab and add **0A** (zero A) after each **0D** (zero D), and add an additional **0D 0A** at the end of the message body. For example, copy and past the following text into the **HEX** input tab:
47 45 54 20 2F 69 70 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F 73 74 3A 20 68 74 74 70 62 69 6E
2E 6F 72 67 0D 0A 0D 0A

Note The HTTP protocol requires an empty line (a line with nothing preceding the CRLF) to terminate the request.

15. Click **Add packet**.
16. Click the **Open** button .

17. Click **Send selected packet**.
18. A GET HTTP response from httpbin.org appears in the Console log.

Debugging

If you experience problems with the settings in this example, you can load the default settings in XCTU:

1. On the Configuration toolbar, click the **Default** button  to load the default values established by the firmware, and click **Yes** to confirm.
2. Factory settings are loaded but not written to the device. To write them, click the **Write** button  on the toolbar.

Get started with MQTT

MQ Telemetry Transport (MQTT) is a messaging protocol that is ideal for the Internet of Things (IoT) due to a light footprint and its use of the publish-subscribe model. In this model, a client connects to a broker, a server machine responsible for receiving all messages, filtering them, and then sending messages to the appropriate clients.

The first two MQTT examples do not involve the XBee Smart Modem. They demonstrate using the MQTT libraries because those libraries are required for [Use MQTT over the XBee Cellular Modem with a PC](#).

The examples in this guide assume:

- Some knowledge of Python.
- An integrated development environment (IDE) such as PyCharm, IDLE or something similar.

The examples require:

- An XBee Smart Modem.
- A compatible development board, such as the XBIB-U.
- XCTU. See [Configure the device using XCTU](#).
- That you install Python on your computer. You can download Python from: <https://www.python.org/downloads/>.
- That you install the **pyserial** and **paho-mqtt** libraries to the Python environment. If you use Python 2, install these libraries from the command line with **pip install pyserial** and **pip install paho-mqtt**. If you use Python 3, use **pip3 install pyserial** and **pip3 install paho-mqtt**.
- The full MQTT library source code, which includes examples and tests, which is available in the paho-mqtt github repository at <https://github.com/eclipse/paho.mqtt.python>. To download this repository you must have Git installed.

Example: MQTT connect

This example provides insight into the structure of packets in MQTT as well as the interaction between the client and broker. MQTT uses different packets to accomplish tasks such as connecting, subscribing, and publishing. You can use XCTU to perform a basic example of sending a broker a connect packet and receiving the response from the server, without requiring any coding. This is a good way to see how the client interacts with the broker and what a packet looks like. The following table is an example connect packet:

	Description	Hex value
CONNECT packet fixed header		
byte 1	Control packet type	0x10
byte 2	Remaining length	0x10
CONNECT packet variable header		
Protocol name		

	Description	Hex value
byte 1	Length MSB (0)	0x00
byte 2	Length LSB (4)	0x04
byte 3	(M)	0x4D
byte 4	(Q)	0x51
byte 5	(T)	0x54
byte 6	(T)	0x54
Protocol level		
byte 7	Level (4)	0x04
Connect flags		
byte 8	CONNECT flags byte, see the table below for the bits.	0X02
Keep alive		
byte 9	Keep Alive MSB (0)	0X00
byte 10	Keep Alive LSB (60)	0X3C
Client ID		
byte 11	Length MSB (0)	0x00
byte 12	Length LSB (4)	0x04
byte 13	(D)	0x44
byte 14	(I)	0x49
byte 15	(G)	0x47
byte 16	(I)	0x49

The following table describes the fields in the packet:

Field name	Description
Protocol Name	The connect packet starts with the protocol name, which is MQTT. The length of the protocol name (in bytes) is immediately before the name itself.
Protocol Level	Refers to the version of MQTT in use, in this case a value of 4 indicates MQTT version 3.1.1.
Connect Flags	Indicate certain aspects of the packet. For simplicity, this example only sets the Clean Session flag, which indicates to the client and broker to discard any previous session and start a new one.
Keep Alive	How often the client pings the broker to keep the connection alive; in this example it is set to 60 seconds.

Field name	Description
Client ID	The length of the ID (in bytes) precedes the ID itself. Each client connecting to a broker must have a unique client ID. In the example, the ID is DIGI. When using the Paho MQTT Python libraries, a random alphanumeric ID is generated if you do not specify an ID.

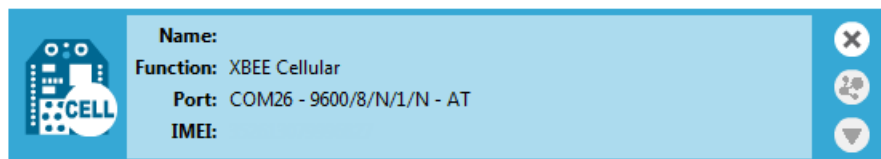
The following table provides the CONNECT flag bits from byte 8, the CONNECT flags byte.

CONNECT Flag Bit(s)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
User name flag	0							
Password flag		0						
Will retain			0					
Will QoS				0	0			
Will flag						0		
Clean session							1	
Reserved								0

Send a connect packet

Now that you know what a connect packet looks like, you can send a connect packet to a broker and view the response. Open XCTU and click the Configuration working mode button.

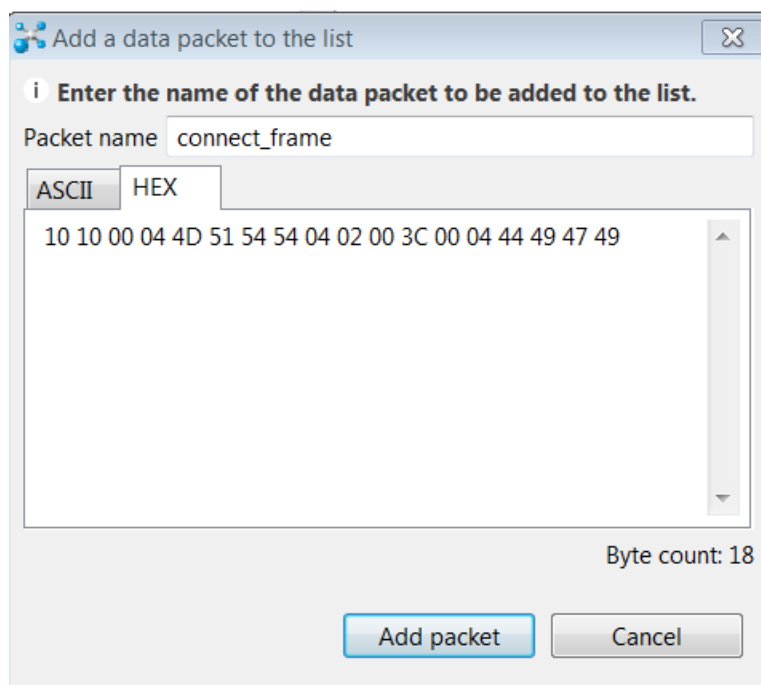
1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and click the **Configuration working mode**  button.
3. Add the XBee Smart Modem to XCTU; see [Add a device](#).
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.



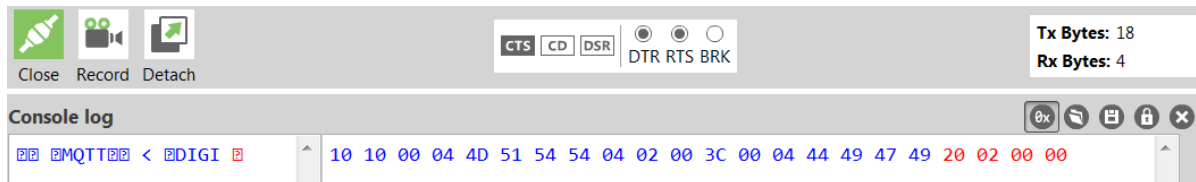
5. In the **AP** field, set **Transparent Mode** to **[0]** if it is not already and click the **Write** button.
6. In the **DL** field, type the IP address of the broker you wish to use. This example uses **198.41.30.241**, which is the IP address for m2m.eclipse.org, a public MQTT broker.
7. In the **DE** field, type **75B** and set the port that the broker uses. This example uses **75B**, because the default MQTT port is 1883 (0x75B).
8. Once you have entered the required values, click the **Write** button to write the changes to the XBee Smart Modem.

9. Click the **Consoles working mode** button  on the toolbar to open a serial console to the device. For instructions on using the Console, see the [AT console](#) topic in the *XCTU User Guide*.
10. Click the **Open** button  to open a serial connection to the device.
11. From the AT console, click the **Add new packet button**  in the Send packets dialog. The **Add new packet** dialog appears.
12. Enter the name of the data packet. Name the packet **connect_frame** or something similar.
13. Click the **HEX** input tab and type the following (these values are the same values from the table in [Example: MQTT connect](#)):

10 10 00 04 4D 51 54 54 04 02 00 3C 00 04 44 49 47 49



14. Click **Add packet**. The new packet appears in the Send packets list.
15. Click the packet in the **Send packets** list.
16. Click **Send selected packet**.
17. A CONNACK packet response from the broker appears in the **Console log**. This is a connection acknowledgment; a successful response should look like this:



You can verify the response from the broker as a CONNACK by comparing it to the structure of a CONNACK packet in the MQTT documentation, which is available at http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718081).

Example: send messages (publish) with MQTT

A basic Python example of a node publishing (sending) a message is:

```

mqttc = mqtt.Client("digitest") # Create instance of client with client ID
                                "digitest"
mqttc.connect("m2m.eclipse.org", 1883) # Connect to (broker, port, keepalive-
time)
mqttc.loop_start() # Start networking daemon
mqttc.publish("digitest/test1", "Hello, World!") # Publish message to "digitest
/test1" topic
mqttc.loop_stop() # Kill networking daemon

```

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

This example imports the MQTT library, allowing you to use the MQTT protocol via APIs in the library, such as the **connect()**, **subscribe()**, and **publish()** methods.

The second line creates an instance of the client, named **mqttc**. The client ID is the argument you passed in: **digitest** (this is optional).

In line 3, the client connects to a public broker, in this case **m2m.eclipse.org**, on port **1883** (the default MQTT port, or 8883 for MQTT over SSL). There are many publicly available brokers available, you can find a list of them here: <https://github.com/mqtt/mqtt.github.io/wiki/brokers>.

Line 4 starts the networking daemon with **client.loop_start()** to handle the background network/data tasks.

Finally, the client publishes its message **Hello, World!** to the broker under the topic **digitest/backlog/test1**. Any nodes (devices, phones, computers, even microcontrollers) subscribed to that same topic on the same broker receive the message.

Once no more messages need to be published, the last line stops the network daemon with **client.loop_stop()**.

Example: receive messages (subscribe) with MQTT

This example describes how a client would receive messages from within a specific topic on the broker:

```

import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc): # The callback for when the client
connects to the broker
    print("Connected with result code {}".format(str(rc))) # Print result of
connection attempt
    client.subscribe("digitest/test1") # Subscribe to the topic
    "digitest/test1", receive any messages published on it

def on_message(client, userdata, msg): # The callback for when a PUBLISH message
is received from the server.
    print("Message received-> " + msg.topic + " " + str(msg.payload)) # Print a

```

```
received msg
```

```
client = mqtt.Client("digi_mqtt_test") # Create instance of client with client
ID "digi_mqtt_test"
client.on_connect = on_connect # Define callback function for successful
connection
client.on_message = on_message # Define callback function for receipt of a
message
# client.connect("m2m.eclipse.org", 1883, 60) # Connect to (broker, port,
keepalive-time)
client.connect('127.0.0.1', 17300)
client.loop_forever() # Start networking daemon
```

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

The first line imports the library functions for MQTT.

The functions **on_connect** and **on_message** are callback functions which are automatically called by the client upon connection to the broker and upon receiving a message, respectively.

The **on_connect** function prints the result of the connection attempt, and performs the subscription. It is wise to do this in the callback function as it guarantees the attempt to subscribe happens only after the client is connected to the broker.

The **on_message** function prints the received message when it comes in, as well as the topic it was published under.

In the body of the code, we:

- Instantiate a client object with the client ID **digi_mqtt_test**
- Define the callback functions to use upon connection and upon message receipt
- Connect to an MQTT broker at **m2m.eclipse.org**, on port **1883** (the default MQTT port, or 8883 for MQTT over SSL) with a keepalive of 60 seconds (this is how often the client pings the broker to keep the connection alive).

The last line starts a network daemon that runs in the background and handles data transactions and messages, as well as keeping the socket open, until the script ends.

Use MQTT over the XBee Cellular Modem with a PC

To use this MQTT library over an XBee Smart Modem, you need a basic proxy that transfers a payload received via the MQTT client's socket to the serial or COM port that the XBee Smart Modem is active on, as well as the reverse; transfer of a payload received on the XBee Smart Modem's serial or COM port to the socket of the MQTT client. This is simplest with the XBee Smart Modem in Transparent mode, as it does not require code to parse or create API frames, and not using API frames means there is no need for them to be queued for processing.

1. To put the XBee Cellular Modem in Transparent mode, set **AP** to **0**.
2. Set **DL** to the IP address of the broker you want to use.
3. Set **DE** to the port to use, the default is 1883 (0x75B). This sets the XBee Smart Modem to communicate directly with the broker, and can be performed in XCTU as described in [Example: MQTT connect](#).
4. You can make the proxy with a dual-threaded Python script, a simple version follows:

```

import threading
import serial
import socket

def setup():
    """
    This function sets up the variables needed, including the serial port,
    and it's speed/port settings, listening socket, and localhost address.
    """
    global clisock, cliaddr, svrsock, ser
    # Change this to the COM port your XBee Cellular module is using. On
    # Linux, this will be /dev/ttyUSB#
    comport = 'COM44'
    # This is the default serial communication speed of the XBee Cellular
    # module
    comspeed = 115200
    buffer_size = 4096 # Default receive size in bytes
    debug_on = 0 # Enables printing of debug messages
    toval = None # Timeout value for serial port below
    # Serial port object for XBCell modem
    ser = serial.Serial(comport,comspeed,timeout=toval)
    # Listening socket (accepts incoming connection)
    svrsock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    # Allow address reuse on socket (eliminates some restart errors)
    svrsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    clisock = None
    cliaddr = None # These are first defined before thread creation
    addrtuple = ('127.0.0.1', 17300) # Address tuple for localhost
    # Binds server socket to localhost (allows client program connection)
    svrsock.bind(addrtuple)
    svrsock.listen(1) # Allow (1) connection

def ComReaderThread():
    """
    This thread listens on the defined serial port object ('ser') for data
    from the modem, and upon receipt, sends it out to the client over the
    client socket ('clisock').
    """
    global clisock
    while (1):
        resp = ser.read() ## Read any available data from serial port
        print("Received {} bytes from modem.".format(len(resp)))

        clisock.sendall(resp) # Send RXd data out on client socket
        print("Sent {} byte payload out socket to client.".format(len(resp)))

def SockReaderThread():
    """
    This thread listens to the MQTT client's socket and upon receiving a
    payload, it sends this data out on the defined serial port ('ser') to the
    modem for transmission.
    """
    global clisock
    while (1):
        data = clisock.recv(4096) # RX data from client socket

```

```

    # If the RECV call returns 0 bytes, the socket has closed
    if (len(data) == 0):
        print("ERROR - socket has closed. Exiting socket reader thread.")
        return 1 # Exit the thread to avoid a loop of 0-byte receptions
    else:
        print("Received {} bytes from client via socket.".format(len(data)))
        print("Sending payload to modem...")
        bytes_wr = ser.write(data) # Write payload to modem via UART/serial
        print("Wrote {} bytes to modem".format(bytes_wr))

def main():
    setup() # Setup the serial port and socket
    global clisock, svrsock
    if (not clisock): # Accept a connection on 'svrsock' to open 'clisock'
        print("Awaiting ACCEPT on server sock...")
        (clisock, cliaddr) = svrsock.accept() # Accept an incoming connection
        print("Connection accepted on socket")
    # Make thread for ComReader
    comthread = threading.Thread(target=ComReaderThread)
    comthread.start() # Start the thread
    # Make thread for SockReader
    sockthread = threading.Thread(target=SockReaderThread)
    sockthread.start() # Start the thread

main()

```

Note This script is a general TCP-UART proxy, and can be used for other applications or scripts that use the TCP protocol. Its functionality is not limited to MQTT.

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

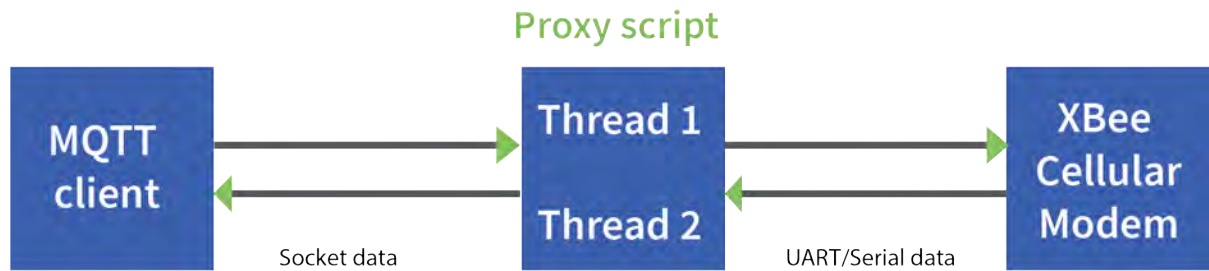
This proxy script waits for an incoming connection on localhost (**127.0.0.1**), on port **17300**. After accepting a connection, and creating a socket for that connection (**clisock**), it creates two threads, one that reads the serial or COM port that the XBee Smart Modem is connected to, and one that reads the socket (**clisock**), that the MQTT client is connected to.

With:

- The proxy script running
- The MQTT client connected to the proxy script via localhost (**127.0.0.1**)
- The XBee Smart Modem connected to the machine via USB and properly powered
- **AP**, **DL**, and **DE** set correctly

the proxy acts as an intermediary between the MQTT client and the XBee Smart Modem, allowing the MQTT client to use the data connection provided by the device.

Think of the proxy script as a translator between the MQTT client and the XBee Smart Modem. The following figure shows the basic operation.



The thread that reads the serial port forwards any data received onward to the client socket, and the thread reading the client socket forwards any data received onward to the serial port. This is represented in the figure above.

The proxy script needs to be running before running an MQTT publish or subscribe script.

1. With the proxy script running, run the subscribe example from [Example: receive messages \(subscribe\) with MQTT](#), but change the connect line from `client.connect("m2m.eclipse.org", 1883, 60)` to `client.connect("127.0.0.1", port=17300, keepalive=20)`. This connects the MQTT client to the proxy script, which in turn connects to a broker via the XBee Smart Modem's internet connection.
2. Run the publish example from [Example: send messages \(publish\) with MQTT](#) in a third Python instance (while the publish script is running you will have three Python scripts running at the same time).

The publish script runs over your computer's normal internet connection, and does not use the XBee Smart Modem. You are able to see your published message appear in the subscribe script's output once it is received from the broker via the XBee Smart Modem. If you watch the output of the proxy script during this process you can see the receptions and transmissions taking place.

The proxy script must be running before you run the subscribe and publish scripts. If you stop the subscribe script, the socket closes, and the proxy script shows an error. If you try to start the proxy script after starting the subscribe script, you may also see a socket error. To avoid these errors, it is best to start the scripts in the correct order: proxy, then subscribe, then publish.

Get started with CoAP

Constrained Application Protocol (CoAP) is based on UDP connection and consumes low power to deliver similar functionality to HTTP. This guide contains information about sending GET, POST, PUT and DELETE operations by using the Coap Protocol with XCTU and Python code working with the XBee Smart Modem and Coapthon library (Python 2.7 only).

The Internet Engineering Task Force describes CoAP as:

The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation. CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments ([source](#)).

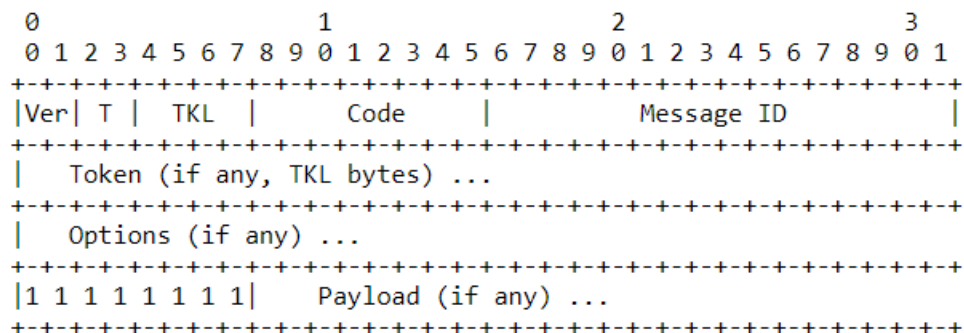
CoAP terms

When describing CoAP, we use the following terms:

Term	Meaning
Method	COAP's method action is similar to the HTTP method. This guide discusses the GET, POST, PUT and DELETE methods. With these methods, the XBee Smart Modem can transport data and requests.
URI	URI is a string of characters that identifies a resource served at the server.
Token	A token is an identifier of a message. The client uses the token to verify if the received message is the correct response to its query.
Payload	The message payload is associated with the POST and PUT methods. It specifies the data to be posted or put to the URI resource
Message ID	The message ID is also an identifier of a message. The client matches the message ID between the response and query.

CoAP quick start example

The following diagram shows the message format for the CoAP protocol; see [ISSN: 2070-1721](#) for details:



This is an example GET request:

```
44 01 C4 09 74 65 73 74 B7 65 78 61 6D 70 6C 65
```

The following table describes the fields in the GET request.

Field	HEX	Bits	Meaning
Ver	44	01	Version 01, which is mandatory here.
T		00	Type 0: confirmable.
TKL		0100	Token length: 4.
Code	01	000 00001	Code: 0.01, which indicates the GET method.
Message ID	C4 09	2 Bytes equal to hex at left	Message ID. The response message will have the same ID. This can help out identification.
Token	74 65 73 74	4 Bytes equal to hex at left	Token. The response message will have the same token. This can help out identification.
Option delta	B7	1011	Delta option: 11 indicates the option data is Uri-Path.
Option length		0111	Delta length: 7 indicates there are 7 bytes of data following as a part of this delta option.
Option value	65 78 61 6D 70 6C 65	7 Bytes equal to hex at left	Example.

Configure the device

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in [Connect the hardware](#).
2. Open XCTU and click the **Configuration working mode**  button.
3. Add the XBee Smart Modem to XCTU; see [Add a device](#).
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. To switch to UDP communication, in the **IP** field, select **0** and click the **Write** button .
6. To set the target IP address that the XBee Smart Modem will talk to, in the **DL** field type **52.43.121.77** and click the **Write** button . A CoAP server is publicly available at address 52.43.121.77.
7. To set the XBee Smart Modem to send data to port 5683 in decimal, in the **DE** field, type **1633** and click the **Write** button.
8. To move into Transparent mode, in the **AP** field, select **0** and click the **Write** button.
9. Wait for the **AI** (Association Indication) value to change to **0** (Connected to the Internet). You can click **Read**  to get an update on the **AI** value.

Example: manually perform a CoAP request

Follow the steps in [Configure the device](#) prior to this example. This example performs the CoAP GET request:

- Method: GET
 - URI: example
 - Given message token: test
1. Click the **Consoles working mode** button  on the toolbar to add a customized packet.
 2. From the AT console, click the **Add new packet button**  in the Send packets dialog. The **Add new packet** dialog appears.
 3. Click the **HEX** tab and type the name of the data packet: **GET_EXAMPLE**.
 4. Copy and past the following text into the **HEX** input tab:
44 01 C4 09 74 65 73 74 B7 65 78 61 6D 70 6C 65
This is the CoAP protocol message decomposed by bytes to perform a GET request on an example URI with a token test.
 5. Click **Add packet**.
 6. Click the **Open** button .
 7. Click **Send selected packet**. The message is sent to the public CoAP server configured in [Configure the device](#). A response appears in the Console log. Blue text is the query, red text is the response.

The payload is **Get to uri: example**, which specifies that this is a successful CoAP GET to URI end example, which was specified in the query.

Click the **Close** button to terminate the serial connection.

Example: use Python to generate a CoAP message

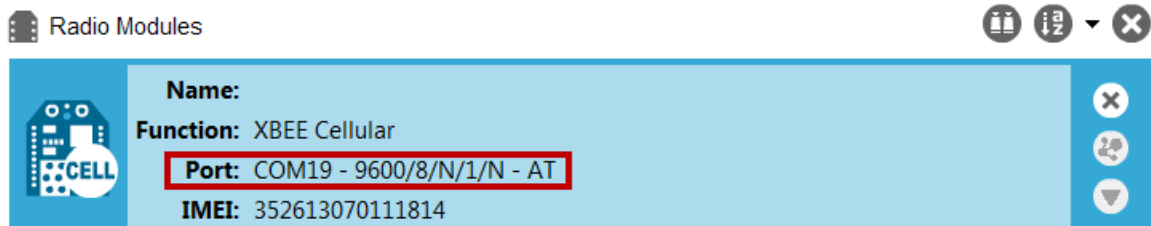
This example illustrates how the CoAP protocol can perform GET/POST/PUT/DELETE requests similarly to the HTTP protocol and how to do this using the XBee Smart Modem. In this example, the XBee Smart Modem talks to a CoAP Digi Server. You can use this client code to provide an abstract wrapper to generate a CoAP message that commands the XBee Smart Modem to talk to the remote CoAP server.

Note It is crucial to configure the XBee Smart Modem settings. See [Configure the device](#) and follow the steps. You can target the IP address to a different CoAP public server.

1. Install Python 2.7. The Installation guide is located at: <https://www.python.org/downloads/>.
2. Download and install the Coapthon library in the python environment from <https://pypi.python.org/pypi/CoAPthon>.
3. Download these two .txt files: [Coap.txt](#) and [CoapParser.txt](#). After you download them, open the files in a text editor and save them as .py files.
4. In the folder that you place the Coap.py and CoapParser.py files, press **Shift + right-click** and then click **Open command window**.
5. At the command prompt, type **python Coap.py** and press **Enter** to run the program.

6. Type the USB port number that the XBee Smart Modem is connected to and press **Enter**. Only the port number is required, so if the port is COM19, type 19.

Note If you do not know the port number, open XCTU and look at the XBee Smart Modem in the **Radio Modules** list. This view provides the port number and baud rate, as in the figure below where the baud rate is 9600 b/s.



7. Type the baud rate and press **Enter**. You must match the device's current baud rate. XCTU provides the current baud rate in the **BD Baud Rate** field. In this example you would type **9600**.
8. Press **Y** if you want an auto-generated example. Press **Enter** to build your own CoAP request.
9. If you press **Y** it generates a message with:
 - Method: POST
 - URI: example
 - payload: hello world
 - token: test

The send and receive message must match the same token and message id. Otherwise, the client re-attempts the connection by sending out the request.

In the following figure, the payload contains the server response to the query. It shows the results for when you press **Enter** rather than **Y**.

```
C:\Users\jzhang\Desktop\example>python Coap.py
Please enter the serial port number for Xbee: 18
Please enter the baudrate number of Xbee: <9600 or 115200>: 9600
Do you want an auto-generated example <Press Y> or build your own <Press ENTER>:

Please enter the HTTP method <GET, POST, PUT, DELETE>: PUT
Please enter the uri end path: example
Please enter the payload content. And it cannot be empty: hello world
Please enter the token: digi

#####

This is the send out message:
Source: <None, None>
Destination: None
Type: CON
MID: 56045
Code: PUT
Token: digi
Uri-Path: example
Payload:
hello world

This is the received message
Source: <None, None>
Destination: None
Type: ACK
MID: 56045
Code: CHANGED
Token: digi
Payload:
Put hello world to uri: example
```

Configure the XBee Smart Modem using Digi Remote Manager

Use Digi Remote Manager (<https://remotemanager.digi.com/>) to perform the operations in this section. Each operation requires that you enable Remote Manager with the **DO** command and that you connect the XBee Smart Modem to an access point that has an external Internet connection to allow access to Digi Remote Manager.

Note Digi is consolidating our cloud services, Digi Device Cloud and Digi Remote Manager®, under the Remote Manager name. This phased process does not affect device functionality or the functionality of the web services and other features. However, customers will find that some user interface and firmware functionality mention both Device Cloud and Digi Remote Manager.

Create a Remote Manager account

Digi Remote Manager is an on-demand service with no infrastructure requirements. Remote devices and enterprise business applications connect to Remote Manager through standards-based web services. This section describes how to configure and manage an XBee using Remote Manager. For detailed information on using Remote Manager, refer to the Remote Manager User Guide, available via the Documentation tab in Remote Manager.

Before you can manage an XBee with Remote Manager, you must create a Remote Manager account. To create a Remote Manager account:

1. Go to <https://www.digi.com/products/cloud/digi-remote-manager>.
2. Click **30 DAY FREE TRIAL/LOGIN**.
3. Follow the online instructions to complete account registration. You can upgrade your Developer account to a paid account at any time.

When you are ready to deploy multiple XBee Smart Modems in the field, upgrade your account to access additional Remote Manager features.

Get the XBee Smart Modem IMEI number

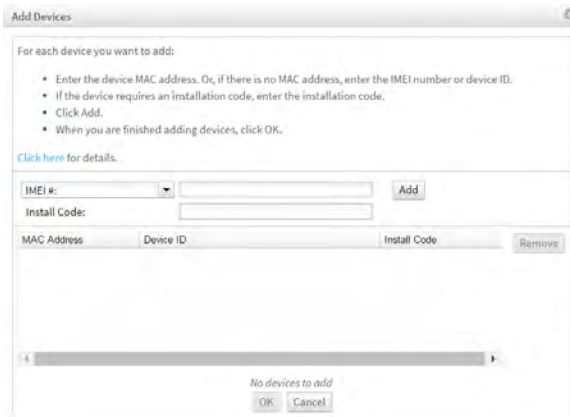
Before adding an XBee to your Remote Manager account inventory, you need to determine the International Mobile Equipment Identity (IMEI) number for the device. Use XCTU to view the IMEI number by querying the **IM** parameter.

Add a XBee Smart Modem to Remote Manager

To add an XBee to your Remote Manager account inventory, follow these steps:

Go to <https://remotemanager.digi.com/>.

1. Log in to your account
2. Click **Device Management > Devices**.
3. Click **Add Devices**. The Add Devices dialog appears.
4. Select **IMEI #**, and type or paste the IMEI number of the XBee you want to add. The **IM (IMEI)** command provides this number.



5. Click **Add** to add the device. The XBee is added to your inventory.
6. Click **OK** to close the Add Devices dialog and return to the Devices view.

Update the firmware

XBee Smart Modem supports Remote Manager firmware updates. To perform a firmware update, use the following steps.

1. Download the updated firmware file for your device from [Digi's support site](#). This is a zip file containing .ebin and .mxi files for import.
2. Unzip the file.
3. In your Remote Manager account, click **Device Management > Devices**.
4. Select the first device you want to update.
5. To select multiple devices (must be of the same type), press the Control key and select additional devices.
6. Click **More** in the Devices toolbar and select **Update Firmware** from the Update category of the More menu. The Update Firmware dialog appears.
7. Click **Browse** to select the .ebin file that you unzipped earlier.
8. Click **Update Firmware**. The updated devices automatically reboot when the updates are complete.

Software libraries

One way to communicate with the XBee device is by using a software library. The libraries available for use with the XBee Smart Modem include:

- [XBee Java library](#)
- [XBee Python library](#)

The XBee Java Library is a Java API. The package includes the XBee library, its source code and a collection of samples that help you develop Java applications to communicate with your XBee devices. The XBee Python Library is a Python API that dramatically reduces the time to market of XBee projects developed in Python and facilitates the development of these types of applications, making it an easy process.

Get started with MicroPython

This guide provides an overview of how to use MicroPython with the XBee Smart Modem. For in-depth information and more complex code examples, refer to the [Digi MicroPython Programming Guide](#). Continue with this guide for simple examples to get started using MicroPython on the XBee Smart Modem.

About MicroPython	47
MicroPython on the XBee Smart Modem	47
Use XCTU to enter the MicroPython environment	47
Use the MicroPython Terminal in XCTU	48
Example: hello world	48
Example: turn on an LED	48
Example: code a request help button	49
Exit MicroPython mode	54
Other terminal programs	54
Use picocom in Linux	55

About MicroPython

MicroPython is an open-source programming language based on Python 3, with much of the same syntax and functionality, but modified to fit on small devices with limited hardware resources, such as microcontrollers, or in this case, a cellular modem.

Why use MicroPython

MicroPython enables on-board intelligence for simple sensor or actuator applications using digital and analog I/O. MicroPython can help manage battery life. Cryptic readings can be transformed into useful data, excess transmissions can be intelligently filtered out, modern sensors and actuators can be employed directly, and logic can glue inputs and outputs together in an intelligent way.

For more information about MicroPython, see www.micropython.org.

For more information about Python, see www.python.org.

MicroPython on the XBee Smart Modem

The XBee Smart Modem has MicroPython running on the device itself. You can access a MicroPython prompt from the XBee Smart Modem when you install it in an appropriate development board (XBDB or XBIB), and connect it to a computer via a USB cable.

Note MicroPython does not work with SPI.

The examples in this guide assume:

- You have [XCTU](#) on your computer. See [Configure the device using XCTU](#).
- You have a terminal program installed on your computer. We recommend using the [Use the MicroPython Terminal in XCTU](#). This requires XCTU 6.3.7 or higher.
- You have an XBee Smart Modem installed in an appropriate development board such as an XBIB-U-DEV or an XBIB-2.

Note Most examples in this guide require the XBIB-U-DEV board.

- The XBee Smart Modem is connected to the computer via a USB cable and XCTU recognizes it.
- The board is powered by an appropriate power supply, 12 VDC and at least 1.1 A.

Use XCTU to enter the MicroPython environment

To use the XBee Smart Modem in the MicroPython environment:

1. Use XCTU to add the device(s); see [Configure the device using XCTU](#) and [Add a device](#).
2. The XBee Smart Modem appears as a box in the **Radio Modules** information panel. Each module displays identifying information about itself.
3. Click this box to select the device and load its current settings.
4. To set the device's baud rate to 115200 b/s, in the **BD** field select **115200 [7]** and click the **Write** button . We recommend using flow control to avoid data loss, especially when pasting large amounts of code/text.

5. To put the XBee Smart Modem into MicroPython mode, in the **AP** field select **MicroPython REPL [4]** and click the **Write** button .
6. Note what COM port(s) the XBee Smart Modem is using, because you will need this information when you use terminal communication.

Use the MicroPython Terminal in XCTU

You can use the MicroPython Terminal to communicate with the XBee Smart Modem when it is in MicroPython mode.¹ This requires XCTU 6.3.7 or higher. To enter MicroPython mode, follow the steps in [Use XCTU to enter the MicroPython environment](#). To use the MicroPython Terminal:

1. Click the **Tools** drop-down menu  and select **MicroPython Terminal**. The terminal opens.
2. Click **Open**.
3. In the **Select the Serial/USB port** area, click the COM port that the device uses.
4. Verify that the baud rate and other settings are correct.
5. Click **OK**. The **Open** icon changes to **Close** , indicating that the device is properly connected.

You can now type or paste MicroPython code in the terminal.

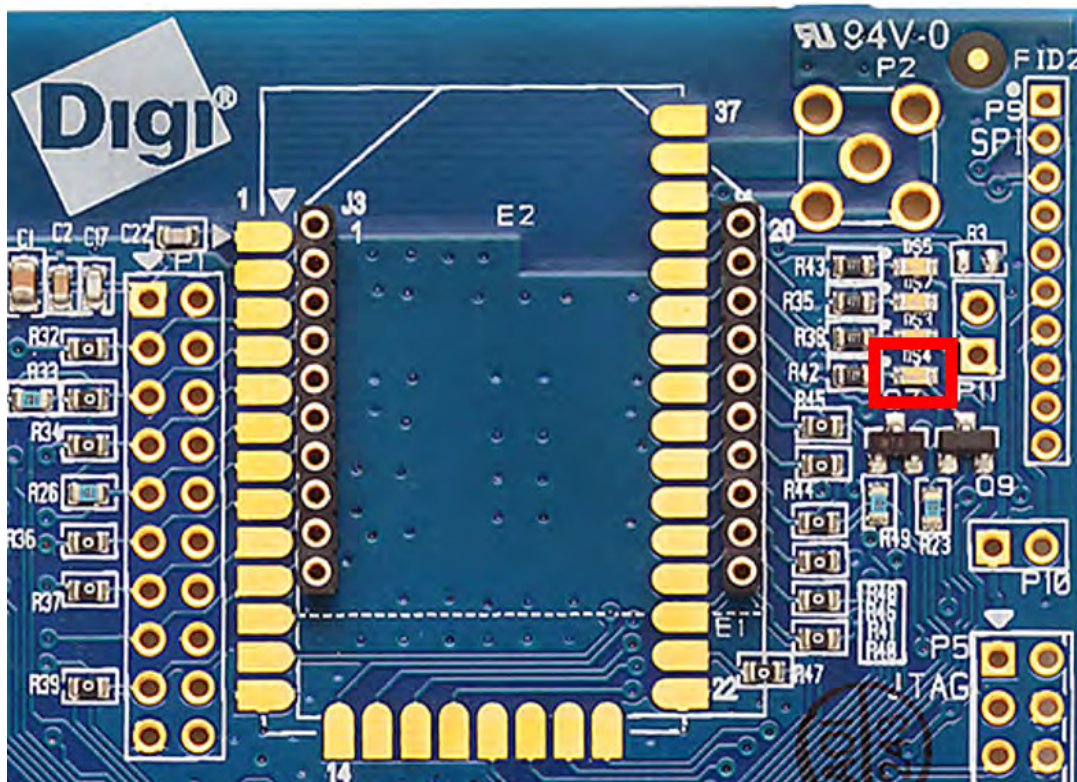
Example: hello world

1. At the MicroPython `>>>` prompt, type the Python command: `print("Hello, World!")`
2. Press **Enter** to execute the command. The terminal echos back **Hello, World!**.

Example: turn on an LED

1. Note the **DS4** LED on the XBIB board. The following image highlights it in a red box. The LED is normally off.

¹See [Other terminal programs](#) if you do not use the MicroPython Terminal in XCTU.



2. At the MicroPython >>> prompt, type the commands below, pressing **Enter** after each one. After entering the last line of code, the LED illuminates. Anything after a # symbol is a comment, and you do not need to type it.

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

```
import machine
from machine import Pin
led = Pin("D4", Pin.OUT, value=0) # Makes a pin object set to output 0.
# One might expect 0 to mean OFF and 1 to mean ON, and this is normally the case.
# But the LED we are turning on and off is setup as what is known as "active
low".
# This means setting the pin to 0 allows current to flow through the LED and then
through the pin, to ground.
```

3. To turn it off, type the following and press **Enter**:

```
led.value(1)
```

You have successfully controlled an LED on the board using basic I/O!

Example: code a request help button

This example provides a fast, deep dive into MicroPython designed to let you see some of the powerful things it can do with minimal code. It is not meant as a tutorial; for in-depth examples refer to the [Digi MicroPython Programming Guide](#).

Many stores have help buttons in their aisles that a customer can press to alert the store staff that assistance is required in that aisle. You can implement this type of system using the Digi XBee Smart Modem, and this example provides the building blocks for such a system. This example, based on SMS paging, can have many other uses such as alerting someone with a text to their phone if a water sensor in a building detects water on the floor, or if a temperature sensor reports a value that is too hot or cold relative to normal operation.

Enter MicroPython paste mode

In the following examples it is helpful to know that MicroPython supports [paste mode](#), where you can copy a large block of code from this user guide and paste it instead of typing it character by character. To use paste mode:

1. Copy the code you want to run. For example, copy the following code that is the code from the LED example:

```
from machine import Pin
led = Pin("D4", Pin.OUT, value=0)
```

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

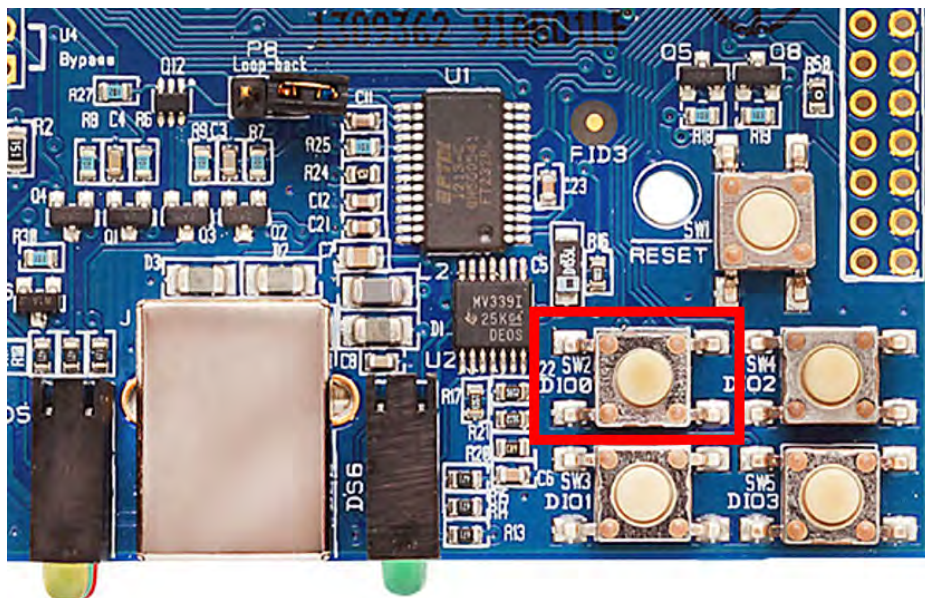
2. In the terminal, at the MicroPython >>> prompt type **Ctrl+E** to enter paste mode. The terminal displays **paste mode; Ctrl-C to cancel, Ctrl-D to finish**.
3. The code appears in the terminal occupying four lines, each line starts with its line number and three = symbols. For example line 1 starts with **1===**.
4. If the code is correct, press **Ctrl+D** to run the code and you should once again see the **DS4** LED turn on. If you get a **Line 1 SyntaxError: invalid syntax** error, see [Syntax error at line 1](#). (If you wish to exit paste mode without running the code, for example, or if the code did not copy correctly, press **Ctrl+C** to cancel and return to the normal MicroPython >>> prompt).
5. Next turn the LED off. Copy the code below:

```
from machine import Pin
led = Pin("D4", Pin.OUT, value=1)
print("DS4 LED now OFF!")
print("Paste Mode Successful!")
```

6. Press **Ctrl+E** to enter paste mode.
7. Press **Ctrl + Shift + V** or right-click in the Terminal and select **Paste** to paste the copied code.
8. If the code is correct, press **Ctrl+D** to run it. The LED should turn off and you should see two confirmation messages print to the screen.

Catch a button press

For this part of the example, you write code that responds to a button press on the XBIB-U-DEV board that comes with the XBee Smart Modem Development Kit. The code monitors the pin connected to the button on the board labeled **SW2**.



On the board you see **DIO0** written below **SW2**, to the left of the button. This represents the pin that the button is connected to.

In MicroPython, you will create a pin object for the pin that is connected to the **SW2** button. When you create the pin object, the **DIO0** pin is called **D0** for short.

The loop continuously checks the value on that pin and once it goes to **0** (meaning the button has been pressed) a **print()** call prints the message **Button pressed!** to the screen.

At the MicroPython **>>>** prompt, copy the following code and enter it into MicroPython using [paste mode](#) and then run it:

```
# Import the Pin module from machine, for simpler syntax.
from machine import Pin

# Create a pin object for the pin that the button "SW2" is connected to.
dio0 = Pin("D0", Pin.IN, Pin.PULL_UP)
# Give feedback to inform user a button press is needed.
print("Waiting for SW2 press...")
# Create a WHILE loop that checks for a button press.
while (True):
    if (dio0.value() == 0): # Once pressed.
        print("Button pressed!") # Print message once pressed.
        break # Exit the WHILE loop.

# When you press SW2, you should see "Button pressed!" printed to the screen.
# You have successfully performed an action in response to a button press!
```

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

Note If you have problems pasting the code, see [Syntax error at line 1](#).

Send a text (SMS) when the button is pressed

After [creating a while loop](#) that checks for a button press, add sending an SMS to your code. Instead of printing **Button pressed!** to the screen, this code sends **Button pressed** to a cell phone as a text (SMS) message.

To accomplish this, use the `sms_send()` method, which sends a string to a given phone number. It takes the arguments in the order of

1. **<phone number>**
2. **<message-to-be-sent>**

Before you run this part of the example, you must create a variable that holds the phone number of the cell phone or mobile device you want to receive the SMS.

1. To do this, at the MicroPython `>>>` prompt, type the following command, replacing **1123456789** with the full phone number (no dashes, spaces, or other symbols) and press **Enter**:

```
ph = 1123456789
```

2. After you create this **ph** variable with your phone number, copy the code below and enter it into MicroPython using [paste mode](#) and then run it.

```
from machine import Pin
import network # Import network module
import time

c = network.Cellular() # initialize cellular network parameter
dio0 = Pin("D0", Pin.IN, Pin.PULL_UP)
while not c.isconnected(): # While no network connection.
    print("Waiting for connection to cell network...")
    time.sleep(5)
print("Connected.")
# Give feedback to inform user a button press is needed.
print("Waiting for SW2 press...")
while (True):
    if (dio0.value() == 0):
        # When SW2 is pressed, the module will send an SMS
        # message saying "Button pressed" to the given target cell phone number.
        try:
            c.sms_send(ph, 'Button Pressed')
            print("Sent SMS successfully.")
        except OSError:
            print("ERROR- failed to send SMS.")
        # Exit the WHILE loop.
        break
```

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

Note If you have problems pasting the code, see [Syntax error at line 1](#). For SMS failures, see [Error Failed to send SMS](#).

Add the time the button was pressed

After you [add the ability to send an SMS](#) to the code, add functionality to insert the time at which the button was pressed into the SMS that is sent. To accomplish this:

1. Create a UDP socket with the **socket()** method.
2. Save the IP address and port of the time server in the **addr** variable.
3. Connect to the time server with the **connect()** method.
4. Send **hello** to the server to prompt it to respond with the current date and time.
5. Receive and store the date/time response in the **buf** variable.
6. Send an SMS in the same manner as before using the **sms_send()** method, except that you add the time into the SMS message, such that the message reads: **[Button pressed at: YYYY-MM-DD HH:MM:SS]**

To verify that your phone number is still in the memory, at the MicroPython **>>>** prompt, type **ph** and press **Enter**.

If MicroPython responds with your number, copy the following code and enter it into MicroPython using [paste mode](#) and then run it. If it returns an error, enter your number again as shown in [Send a text \(SMS\) when the button is pressed](#). With your phone number in memory in the **ph** variable, copy the code below and enter it into MicroPython using [paste mode](#) and then run it.

```
from machine import Pin
import network
import usocket
import time

c = network.Cellular()
dio0 = Pin("D0", Pin.IN, Pin.PULL_UP)
while not c.isconnected(): # While no network connection.
    print("Waiting for connection to cell network...")
    time.sleep(5)
print("Connected.")
# Give feedback to inform user a button press is needed.
print("Waiting for SW2 press...")
while (1):
    if (dio0.value() == 0):
        # When button pressed, now the module will send "Button Press" AND
        # the time at which it was pressed in an SMS message to the given
        # target cell phone number.
        socketObject = usocket.socket(usocket.AF_INET, usocket.SOCK_DGRAM)
        # Connect the socket object to the web server specified in "address".
        addr = ("52.43.121.77", 10002)
        socketObject.connect(addr)
        bytessent = socketObject.send("hello")
        print("Sent %d bytes on socket" % bytessent)
        buf = socketObject.recv(1024)
        # Send message to the given number. Handle error if it occurs.
        try:
            c.sms_send(ph, 'Button Pressed at: ' + str(buf))
            print("Sent SMS successfully.")
        except OSError:
            print("ERROR- failed to send SMS.")
        # Exit the WHILE loop.
        break
```

Note You can easily copy and paste code from the online version of this Guide. Use caution with the PDF version, as it may not maintain essential indentations.

Now you have a system based on the XBee Smart Modem that sends an SMS in response to a certain input, in this case a simple button press.

Note If you have problems pasting the code, see [Syntax error at line 1](#). For SMS failures, see [Error Failed to send SMS](#).

Exit MicroPython mode

To exit MicroPython mode:

1. In the XCTU MicroPython Terminal, click the green **Close** button .
2. Click **Close** at the bottom of the terminal to exit the terminal.
3. In XCTU's Configuration working mode , change **AP API Enable** to another mode and click the **Write** button . We recommend changing to Transparent mode **[0]**, as most of the examples use this mode.

Other terminal programs

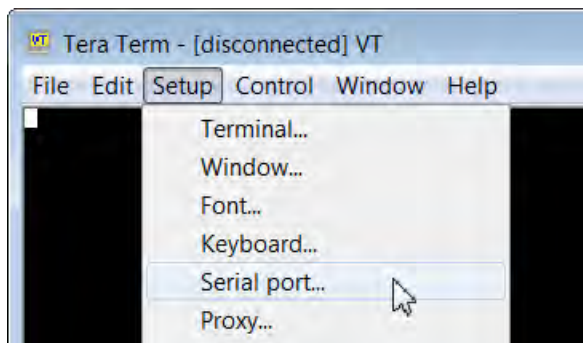
If you do not use the MicroPython Terminal in XCTU, you can use other terminal programs to communicate with the XBee Smart Modem. If you use Microsoft Windows, follow the instructions for Tera Term, if you use Linux, follow the instructions for picocom. To download these programs:

- Tera Term for Windows; see <https://ttssh2.osdn.jp/index.html.en>.
- Picocom for Linux; see https://developer.ridgerun.com/wiki/index.php/Setting_up_Picocom_-_Ubuntu and for the source code and in-depth information <https://github.com/npatefault/picocom>.

Tera Term for Windows

With the XBee Smart Modem in MicroPython mode (**AP = 4**), you can access the MicroPython prompt using a terminal.

1. Open Tera Term. The **Tera Term: New connection** window appears.
2. Click the **Serial** radio button to select a serial connection.
3. From the **Port:** drop-down menu, select the COM port that the XBee Smart Modem is connected to.
4. Click **OK**. The **COMxx - Tera Term VT** terminal window appears and Tera Term attempts to connect to the device at a baud rate of 9600 b/s. The terminal will not allow communication with the device since the baud rate setting is incorrect. You must change this rate as it was previously set to 115200 b/s.
5. Click **Setup** and **Serial Port**. The **Tera Term: Serial port setup** window appears.



6. In the **Tera Term: Serial port setup** window, set the parameters to the following values:
 - **Port:** Shows the port that the XBee Smart Modem is connected on.
 - **Baud rate:** 115200
 - **Data:** 8 bit
 - **Parity:** none
 - **Stop:** 1 bit
 - **Flow control:** hardware
 - **Transmit delay:** N/A
7. Click **OK** to apply the changes to the serial port settings. The settings should go into effect right away.
8. To verify that local echo is not enabled and that extra line-feeds are not enabled:
 - a. In Tera Term, click **Setup** and select **Terminal**.
 - b. In the **New-line** area of the **Tera Term: Serial port setup** window, click the **Receive** drop-down menu and select **CR** if it does not already show that value.
 - c. Make sure the **Local echo** box is not checked.
9. Click **OK**.
10. Press **Ctrl+B** to get the MicroPython version banner and prompt.

```
MicroPython v1.8.7 on 2017-04-06; XBee Cellular with EFM32G
Type "help()" for more information.
>>>
```

Now you can type MicroPython commands at the `>>>` prompt.

Use picocom in Linux

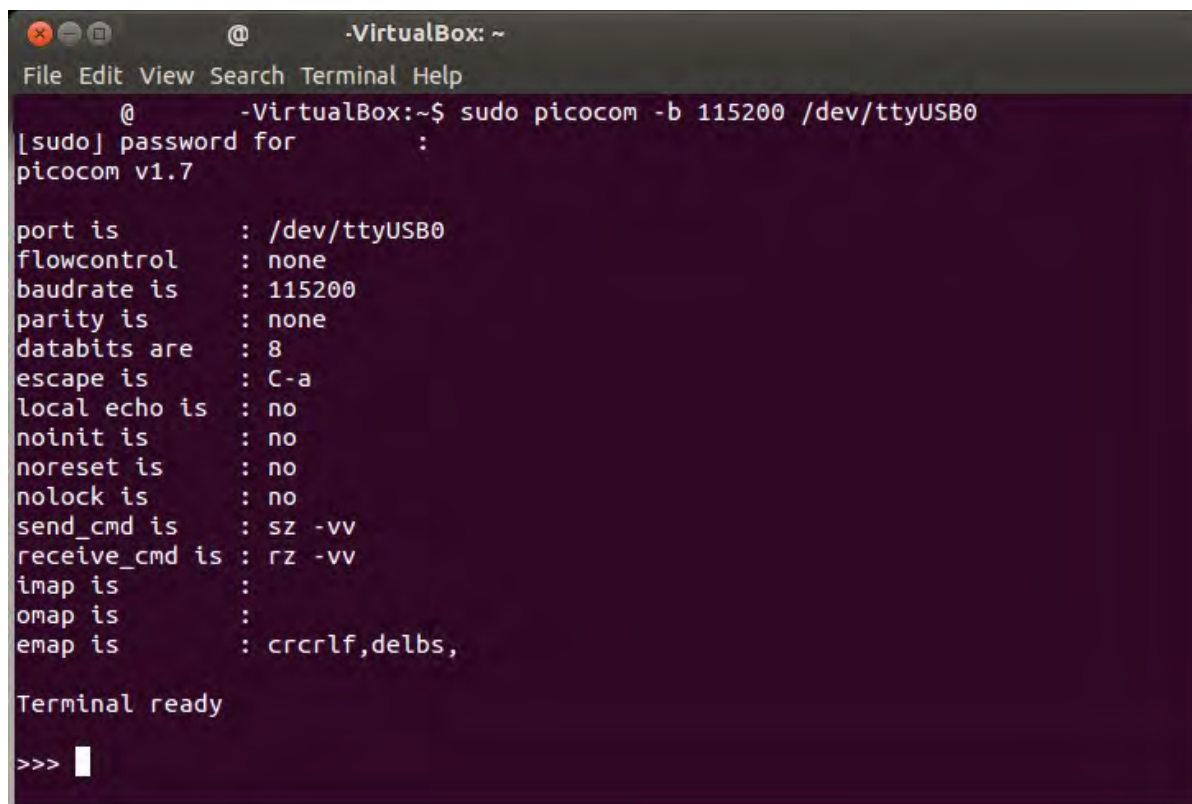
With the XBee Smart Modem in MicroPython mode (**AP = 4**), you can access the MicroPython prompt using a terminal.

Note The user must have read and write permission for the serial port the XBee Smart Modem is connected to in order to communicate with the device.

1. Open a terminal in Linux and type **picocom -b 115200 /dev/ttyUSB0**. This assumes you have no other USB-to-serial devices attached to the system.
2. Press **Ctrl+B** to get the MicroPython version banner and prompt. You can also press **Enter** to bring up the prompt.

If you do have other USB-to-serial devices attached:

1. Before attaching the XBee Smart Modem, check the directory **/dev/** for any devices named **tttyUSBx**, where **x** is a number. An easy way to list these is to type: **ls /dev/ttyUSB***. This produces a list of any device with a name that starts with **tttyUSB**.
2. Take note of the devices present with that name, and then connect the XBee Smart Modem.
3. Check the directory again and you should see one additional device, which is the XBee Smart Modem.
4. In this case, replace **/dev/ttyUSB0** at the top with **/dev/ttyUSB<number>**, where **<number>** is the new number that appeared.
5. It should connect and show Terminal ready.



```
-VirtualBox: ~
File Edit View Search Terminal Help
@ -VirtualBox:~$ sudo picocom -b 115200 /dev/ttyUSB0
[sudo] password for :
picocom v1.7

port is       : /dev/ttyUSB0
flowcontrol   : none
baudrate is   : 115200
parity is     : none
databits are  : 8
escape is     : C-a
local echo is : no
noinit is     : no
noreset is    : no
nolock is     : no
send_cmd is   : SZ -vv
receive_cmd is : rz -vv
imap is       :
omap is       :
emap is       : crcrlf,delbs,

Terminal ready

>>> █
```

Now you can type MicroPython commands at the **>>>** prompt.

Technical specifications

- Interface and hardware specifications58
- Cellular RF characteristics58
- Bluetooth RF characteristics58
- Cellular Networking specifications58
- Power requirements59
- Power consumption59
- Electrical specifications59
- Regulatory approvals60

Interface and hardware specifications

The following table provides the interface and hardware specifications for the device.

Specification	Value
Dimensions	24.38 mm x 32.94 mm (0.960 x 1.297 in)
Weight	5 g (0.18 oz)
Operating temperature	-40 to +85 °C
Antenna connector	Cellular: U.FL Bluetooth: U.FL
Digital I/O	13 I/O lines
ADC	4 10-bit analog inputs
Cellular chipset	u-blox SARA-R410M
Form factor	Digi XBee 20-pin through-hole
SIM size	4FF Nano

Cellular RF characteristics

The following table provides the RF characteristics for the device.

Specification	Value
Transmit power	Up to 23 dBm, Power Class 3
Receive sensitivity	-105 dBm

Bluetooth RF characteristics

The following table provides the Bluetooth RF characteristics for the device.

Specification	Value
Transmit power	Up to 8 dBm
Receive sensitivity, 1 Mb/s data rate	-94 dBm
Receive sensitivity, 2 Mb/s data rate	-90 dBm
Operating frequency band	ISM 2.4 - 2.4835 GHz

Cellular Networking specifications

The following table provides the networking and carrier specifications for the device.

Specification	Value
Carrier and technology	AT&T LTE-M Compatible with other LTE-M carriers, see supported bands
Supported bands	LTE Bands 2, 4, 5, 12
Security	Digi Trustfence™
Data throughput	TBD
Downlink/uplink speeds	Up to 375 kb/s
Duplex mode	Half-duplex

Power requirements

The following table provides the power requirements for the device.

Specification	Value
Supply voltage	3.3 to 4.3 V

Power consumption

Specification	State	Typical current VCC = 3.3 V
Peak transmit current	Bluetooth disabled	550 mA
	Bluetooth enabled	610 mA
Average transmit current	Active TX/RX @ 23 dBm	235 mA
Active mode current	Idle/connected, listening	24 mA
Power save mode current		20 µA
Deep sleep current		10 µA

Electrical specifications

The following table provides the electrical specifications for the XBee Smart Modem.

Symbol	Parameter	Condition	Min	Typical	Max	Units
VCCMAX	Maximum limits of VCC line		0		4.3	V

Symbol	Parameter	Condition	Min	Typical	Max	Units
VDD_IO	Internal supply voltage for I/O		(VCC - 0.15 V) or 3.3 V, whichever is lower	VCC or 3.3 V, whichever is lower	3.3	V
VI	Voltage on any pin		-0.3		VDD_IO + 0.3	V
VIL	Input low voltage				0.3*VDD_IO	V
VIH	Input high voltage		0.7*VDD_IO			V
VOL	Voltage output low	Sinking 3 mA, VCC = 3.3 V			0.2*VDD_IO	V
VOH	Voltage output high	Sourcing 3 mA, VCC = 3.3 V	0.8*VDD_IO			V
I_IN	Input leakage current	High Z state I/O connected to Ground or VDD_IO		0.1	30	nA
RPU	Internal pull-up resistor	Enabled		40		kΩ
RPD	Internal pull-down resistor	Enabled		40		kΩ

Regulatory approvals

The following table provides the regulatory and carrier approvals for the device.

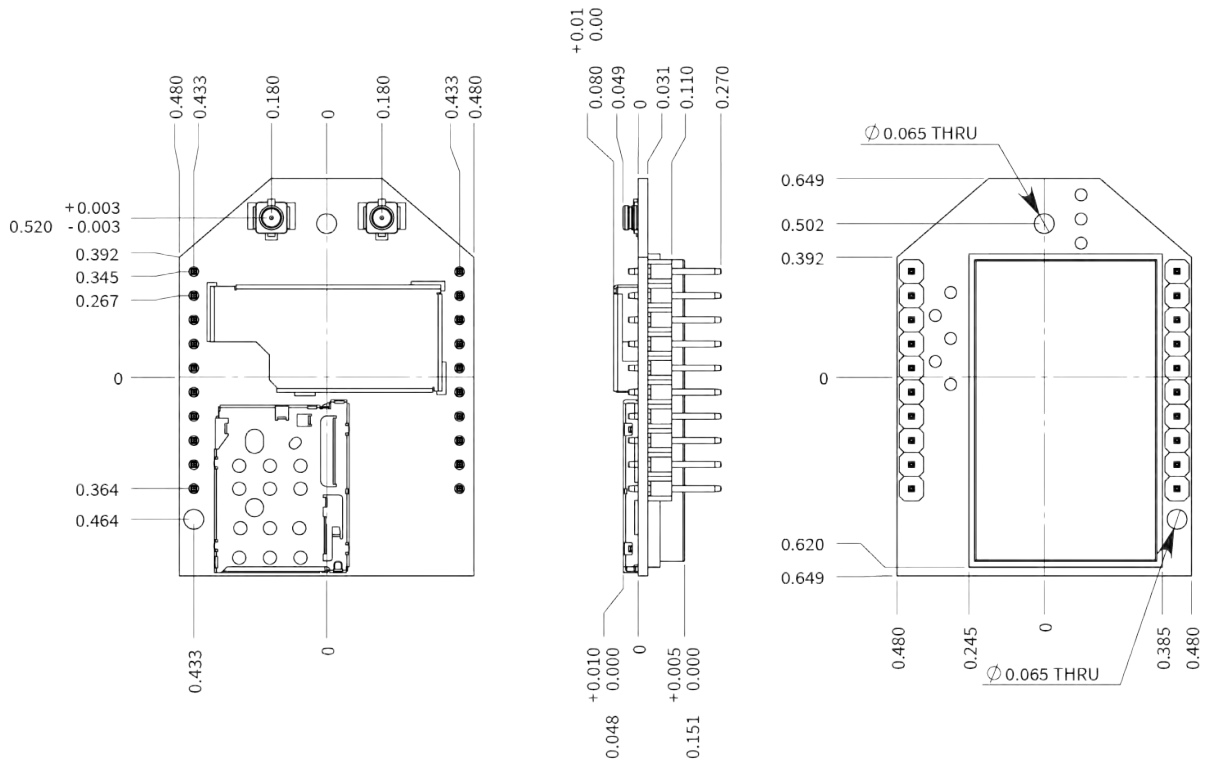
Specification	Value
United States	FCC ID: MCQ-XB3M1 FCC ID: XPY2AGQN4NNN
Industry Canada	IC: 1846A-XB3M1 IC: 8595A-2AGQN4NNN
RoHS	Lead-free and RoHS compliant
AT&T end-device certified	Complete
PTCRB	Complete

Hardware

Mechanical drawings	62
Pin signals	62
RSSI PWM	64
SIM card	64
The Associate LED	65

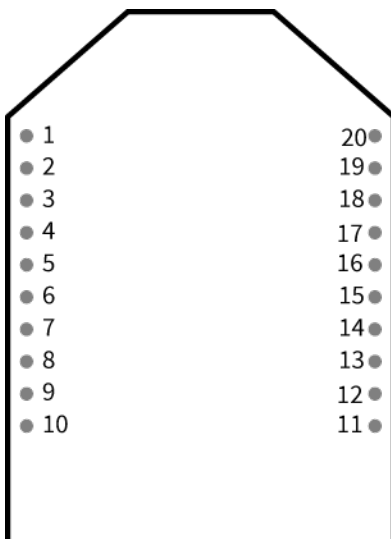
Mechanical drawings

The following figures show the mechanical drawings for the XBee Smart Modem. All dimensions are in inches.



Pin signals

The pin locations are:



The following table shows the pin assignments for the through-hole device. In the table, low-asserted signals have a horizontal line above signal name.

Pin	Name	Direction	Default	Description
Pin	Name	Direction	Default	Description
1	V _{CC}			Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	Input	UART Data In
4	DIO12 / SPI_MISO	Either	Disabled	Digital I/O 12 or SPI Slave Output line
5	$\overline{\text{RESET}}$	Input		
6	PWM0 / RSSI / DIO10/USB_VBUS	Either	Output	PWM Output 0 / RX Signal Strength Indicator / Digital I/O 10 or USB VBUS line ¹
7	DIO11/USB D+	Either	Disabled	Digital I/O 11 or USB D+ line
8	USB D-			Direct USB D- line
9	$\overline{\text{DTR}}$ / SLEEP_RQ/ DIO8	Either	Disabled	Pin Sleep Control Line or Digital I/O 8
10	GND			Ground
11	DIO4 / SPI_MOSI	Either	Disabled	Digital I/O 4 or SPI Slave Input Line
12	$\overline{\text{CTS}}$ / DIO7	Either	Output	Output Clear-to-Send Flow Control or Digital I/O 7
13	ON / $\overline{\text{SLEEP}}$ /DIO9	Output	Output	Module Status Indicator or Digital I/O 9
14	VREF	-		Feature not supported on this device. Used on other XBee devices for analog voltage reference.
15	Associate / DIO5	Either	Output	Associated Indicator, Digital I/O 5
16	$\overline{\text{RTS}}$ / DIO6	Either	Disabled	Input Request-to-Send Flow Control, Digital I/O 6
17	AD3 / DIO3 / SPI_ $\overline{\text{SS}}$	Either	Disabled	Analog Input 3 or Digital I/O 3, SPI low enabled select line

¹The XBee3 device is 5 V tolerant on this pin whereas most other XBee devices are not.

Pin	Name	Direction	Default	Description
18	AD2 / DIO2 / SPI_CLK	Either	Disabled	Analog Input 2 or Digital I/O 2, SPI Clock line
19	AD1 / DIO1 / SPI_ATT $\overline{N}$	Either	Disabled	Analog Input 1 or Digital I/O 1, SPI Attention line output
20	AD0 / DIO0	Either	Input	Analog Input 0, Digital I/O 0

Pin connection recommendations

The recommended minimum pin connections are VCC, GND, DIN, DOUT, $\overline{RTS}$, $\overline{DTR}$ and $\overline{RESET}$. Firmware updates require access to these pins.

RSSI PWM

The XBee Smart Modem features an RSSI/PWM pin (pin 6) that, if enabled, adjusts the PWM output to indicate the signal strength of the cellular connection. Use [P0 \(DIO10/PWM0 Configuration\)](#) to enable the RSSI pulse width modulation (PWM) output on the pin. If **P0** is set to 1, the RSSI/PWM pin outputs a PWM signal where the frequency is adjusted based on the received signal strength of the cellular connection.

The RSSI/PWM output is enabled continuously unlike other XBee products where the output is enabled for a short period of time after each received transmission. If running on the XBIB development board, DIO10 is connected to the RSSI LEDs, which may be interpreted as follows:

PWM duty cycle	Number of LEDs turned on	Received signal strength (dBm)
79.39% or more	3	-83 dBm or higher
62.42% to 79.39%	2	-93 to -83 dBm
45.45% to 62.42%	1	-103 to -93 dBm
Less than 45.45%	0	Less than -103 dBm, or no cellular network connection

SIM card

The XBee Smart Modem uses a 4FF (Nano) size SIM card.



CAUTION! Never insert or remove SIM card while the power is on!

The Associate LED

The following table describes the Associate LED functionality. For the location of the Associate LED on the XBIB-U development board, see number 6 on the [XBIB-U-DEV reference](#).

LED status	Blink timing	Meaning
On, solid		Not joined to a mobile network.
Double blink	½ second	The last TCP/UDP attempt failed. If the LED has this pattern, you may need to check DI (Device Cloud Indicator) or CI (Protocol/Connection Indication) for the cause of the error.
Standard single blink	1 second	Normal operation.

The normal association LED signal alternates evenly between high and low as shown below:



Where the low signal means LED off and the high signal means LED on.

When **CI** is not **0** or **0xFF**, the Associate LED has a different blink pattern that looks like this:



Antenna recommendations

Antenna placement	67
-------------------------	----

Antenna placement

For optimal cellular reception, keep the antenna as far away from metal objects and other electronics (including the XBee Smart Modem) as possible. Often, small antennas are desirable, but come at the cost of reduced range and efficiency.

Design recommendations

- Power supply considerations69
- Minimum connection diagram69
- Heat considerations and testing69
- Heat sink guidelines71
- Add a fan to provide active cooling72

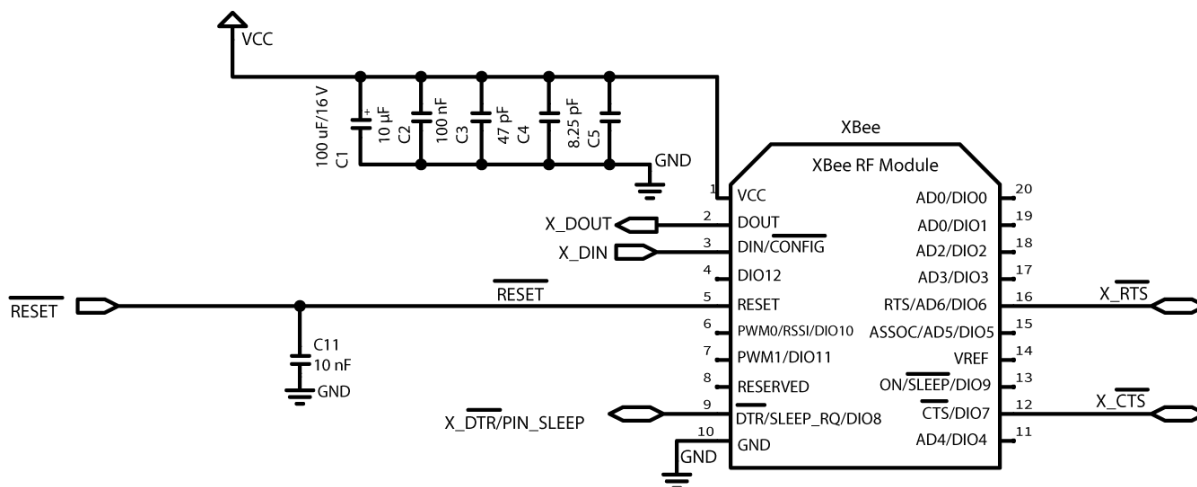
Power supply considerations

When considering a power supply, use the following design practices.

1. Power supply ripple should be less than 75 mV peak to peak.
2. The power supply should be capable of providing a minimum of 750 mA at 3.3 V (2.5 W).
3. Place sufficient bulk capacitance on the XBee VCC pin to maintain voltage above the minimum specification during transmissions. [Power consumption](#) lists the peak current during transmitting.
4. Place smaller high frequency ceramic capacitors very close to the XBee Smart Modem VCC pin to decrease high frequency noise.
5. Use a wide power supply trace or power plane to ensure it can handle the peak current requirements with minimal voltage drop. We recommend that the power supply and trace be designed such that the voltage at the XBee VCC pin does not vary by more than 0.1 V between light load (~0.5 W) and heavy load (~3 W). The supply should be inside the supply voltage operating range at startup and should not be allowed to droop lower than 3.2 V during operation.

Minimum connection diagram

In high EMI noise environments, we recommend adding a 10 nF ceramic capacitor very close to pin 5.



Heat considerations and testing

The XBee Smart Modem may generate significant heat during sustained operation. In addition to heavy data transfer, other factors that can contribute to heating include ambient temperature, air flow around the device, and proximity to the nearest cellular tower (the XBee Smart Modem must transmit at a higher power level when communicating over long distances). Overheating can cause device malfunction and potential damage. In order to avoid this it is important to consider the application the XBee Smart Modem is going into and mitigate heat issues if necessary.

We recommend that you perform thermal testing in your application to determine the resulting steady state temperature of the XBee Smart Modem. Use [TP \(Temperature\)](#) to estimate the device

temperature¹. Convert the **TP** reading from hex format to decimal. We recommend that you confirm the **TP** readings by attaching a thermocouple directly to the onboard microcontroller (if using a heat sink place the thermocouple under the thermal gasket), and reading the temperature from the thermocouple. The location of the microcontroller is shown below.



You also need to know the ambient temperature and the average current consumption during your test. If you do not have a way to measure current consumption you can estimate it from the table in the next section.

Use those results to approximate the maximum safe ambient temperature for the XBee Smart Modem, $T_{MAX,amb}$, with the following equation:

$$T_{MAX,amb} = 80^{\circ}\text{C} - (T_{XBee} - T_{amb,test}) \left(\frac{I_{MAX}}{I_{AVG,test}} \right)$$

Where:

T_{XBee} is the steady state temperature of the XBee Smart Modem that you measured during your test (if using the **TP** command, be sure to convert from hex format to decimal).

$T_{amb,test}$ is the ambient air temperature during your test.

$I_{AVG,test}$ is the average current measured during your test.

I_{MAX} is the maximum current draw expected for your application during transmission.

¹The **TP** reading may not be calibrated. To compensate for this you can determine an offset to use in the equations above as follows: With the XBee Smart Modem not powered, allow it to sit at room temperature for 15 - 20 minutes. Power the device and immediately read the **TP** command. Convert the **TP** reading from hex format to decimal and subtract the result from the actual room temperature. Add this offset to to T_{XBee} in your numbers above.

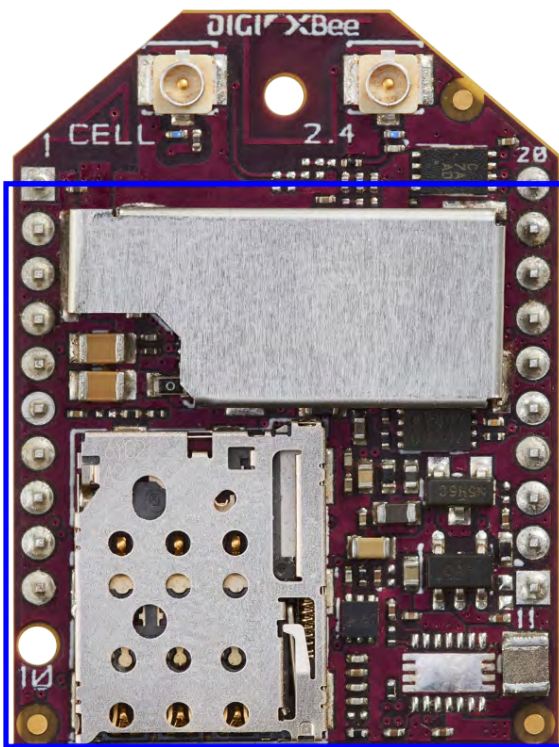
Heat sink guidelines

Based on the results of your thermal testing you may find it is advisable or required to implement a method of dissipating excess heat. This section explains how to employ a heat sink on top of the XBee Smart Modem.

Bolt-down style

A bolt-down style heat sink on top of the XBee Smart Modem provides the best performance. An example part number is Advanced Thermal Solutions ATS-PCBT1084/ATS-PCB1084. You must use an electrically non-conductive thermal gasket on top of the XBee device under the area that will be covered by the heat sink. A thermal gasket such as Gap Pad® 2500S20 is suitable for this purpose. We recommend using a gasket with thickness of 0.080 in to ensure that components on top of the XBee device do not tear through the material when pressure is applied to the heat sink.

Install the SIM card prior to placement of the heat sink. Position the thermal gasket and heat sink assembly on the top of the device so that it covers the microcontroller and surrounding components. You may cover the section shown inside the red box below; do not cover the U.FL connectors. When attaching to the host PCB, tighten the mounting hardware until the thermal gasket is compressed about 25%. Avoid overtightening. To prevent shorting, check that the surface of the heat sink does not directly contact the XBee device.



Adhesive style heat sink

For applications where the size or mounting requirement of the bolt-down heat sink is undesirable, you may alternatively employ an adhesive style heat sink. The heat sink should be no more than 8x8 mm in size (one option is the Assman WSW Components V2016B). Use a thermally conductive epoxy to

attach the heat sink directly to the microcontroller package, and to prevent shorting ensure that the heat sink does not touch any other components.

The following table provides a list of typical scenarios and the maximum ambient temperature at which the XBee Smart Modem can be safely operated under that condition. These are provided only as guidelines as your results will vary based on application. We recommend that you perform sufficient testing, as explained in [Heat considerations and testing](#), to ensure that the XBee Smart Modem does not exceed temperature specifications.

Scenario	Average current consumption (VCC = 3.3 V)	Example application	Peak data consumed (MB/hr)	Maximum ambient temperature			
				No heat sink	Adhesive heat sink	Bolt-down heat sink	Bolt-down heat sink and fan
Sustained operation	235 mA	TBD	TBD	TBD	TBD	TBD	TBD
Device awake, limited transmissions	TBD	Updating traffic sign	1 to 10	TBD	TBD	TBD	85 °C
Device primarily asleep, very limited transmissions	TBD	Small data transmission/receptions which occur once per hour	Less than 0.1	85 °C	85 °C	85 °C	85 °C

Add a fan to provide active cooling

Another option for heat mitigation is to add a fan to your system to provide active cooling. You can use a fan instead of or in addition to a heat sink. The XBee Smart Modem offers a fan control feature on I/O pin DIO11 (pin 7). When the functionality is enabled, that line is pulled high to indicate when the fan should be turned on. The line is pulled high when the device gets above 70 °C and the cellular component is running, and turns off when the device drops below 65 °C.

To enable the functionality set [P1 \(DIO11/PWM1 Configuration\)](#) to **1**. Note that the I/O pin is not capable of driving a fan directly; you must implement a circuit to power the fan from a suitable power source.

Cellular connection process

- Connecting74
- Cellular network74
- Data network connection74
- Data communication with remote servers (TCP/UDP)74
- Disconnecting74

Connecting

In normal operations, the XBee Smart Modem automatically attempts both a cellular network connection and a data network connection on power-up. The sequence of these connections is as follows:

Cellular network

1. The device powers on.
2. It looks for cellular towers.
3. It chooses a candidate tower with the strongest signal.
4. It negotiates a connection.
5. It completes cellular registration.

Data network connection

1. The network enables the evolved packet system (EPS) bearer with an access point name (APN). See [AN \(Access Point Name\)](#) if you have APN issues.
2. The device negotiates a data connection with the access point.
3. The device receives its IP configuration and address.
4. The [AI \(Association Indication\)](#) command now returns a **0** and the sockets become available.

Data communication with remote servers (TCP/UDP)

Once the data network connection is established, communication with remote servers can be initiated in several ways:

- Transparent mode data sent to the serial port (see [TD \(Text Delimiter\)](#) and [RO \(Packetization Timeout\)](#) for timing).
- API mode: [Transmit \(TX\) Request: IPv4 - 0x20](#) received over the serial connection.
- Digi Remote Manager connectivity begins.

Data communication begins when:

1. A socket opens to the remote server.
2. Data is sent.

Data connectivity ends when:

1. The server closes the connection.
2. The **TM** timeout expires (see [TM \(IP Client Connection Timeout\)](#)).
3. The cellular network may also close the connection after a timeout set by the network operator.

Disconnecting

When the XBee Smart Modem is put into Airplane mode or deep sleep is requested:

1. Sockets are closed, cleanly if possible.
2. The cellular connection is shut down.
3. The cellular component is powered off.

Note We recommend entering Airplane mode before resetting or rebooting the device to allow the cellular module to detach from the network.

Modes

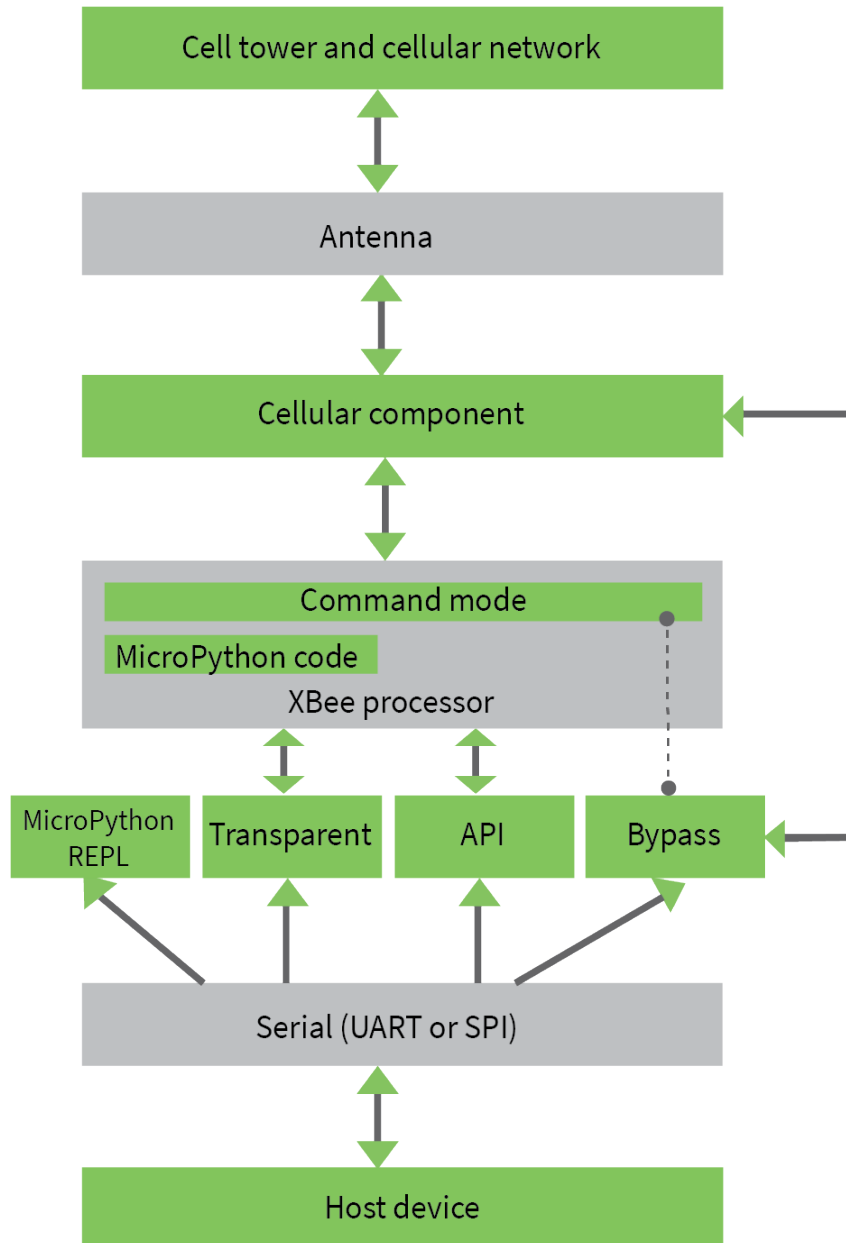
- Select an operating mode 77
- Transparent operating mode 78
- API operating mode 78
- Bypass operating mode 78
- USB direct mode 79
- Command mode 80

Select an operating mode

The XBee Smart Modem interfaces to a host device such as a microcontroller or computer through a logic-level asynchronous serial port. It uses a [UART](#) for serial communication with those devices.

The XBee Smart Modem supports three operating modes: Transparent operating mode, API operating mode, and Bypass operating mode. The default mode is Transparent operating mode. Use the [AP \(API Enable\)](#) command to select a different operating mode.

The following flowchart illustrates how the modes relate to each other.



Transparent operating mode

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

The [IP \(IP Protocol\)](#) command setting controls how Transparent operating mode works for the XBee Smart Modem.

Note Transparent operation is not available when using SPI.

API operating mode

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

The advantages of API operating mode include:

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

Bypass operating mode



CAUTION! Bypass operating mode is an alternative to Transparent and API modes for advanced users with special configuration needs. Changes made in this mode might change or disable the device and we do not recommend it for most users.

In Bypass mode, the device acts as a serial line replacement to the cellular component. In this mode, the XBee Smart Modem exposes all control of the cellular component's AT port through the UART. If you use this mode, you must setup the cellular modem directly to establish connectivity. The modem does not automatically connect to the network.

Note The cellular component can become unresponsive in Bypass mode. See [Unresponsive cellular component in Bypass mode](#) for help in this situation.

When Bypass mode is active, most of the XBee Smart Modem's AT commands do not work. For example, **IM** (IMEI) may never return a value, and **DB** does not update. In this configuration, the firmware does not test communication with the cellular component (which it does by sending AT commands). This is useful in case you have reconfigured the cellular component in a way that makes it incompatible with the firmware. Bypass operating mode exists for users who wish to communicate directly with the cellular component settings and do not intend to use XBee Smart Modem software features such as API mode.

Command mode is available while in Bypass mode; see [Enter Command mode](#) for instructions.

Enter Bypass operating mode

To configure a device for Bypass operating mode:

1. Set the [AP \(API Enable\)](#) parameter value to **5**.
2. Send [WR \(Write\)](#) to write the changes.
3. Send [FR \(Force Reset\)](#) to reboot the device.
4. After rebooting, enter Command mode and verify that Bypass operating mode is active by querying [AI \(Association Indication\)](#) and confirming that it returns a value of **0x2F**.

It may take a moment for Bypass operating mode to become active.

Leave Bypass operating mode

To configure a device to leave Bypass operating mode:

1. Set [AP \(API Enable\)](#) to something other than 5.
2. Send [WR \(Write\)](#) to write the changes.
3. Send [FR \(Force Reset\)](#) to reboot the device.
4. After rebooting, enter Command mode and verify that Bypass operating mode is not active by querying [AI \(Association Indication\)](#) and confirming that it returns a value other than **0x2F**.

Restore cellular settings to default in Bypass operating mode

Send **AT&F1** to reset the cellular component to its factory profile.

USB direct mode

This mode allows you to access the XBee Smart Modem's USB interface directly through XBee pins 7 and 8. VBUS functionality is optionally provided on XBee pin 6 if you wish to enable and disable USB mode based on an external source. While in USB mode the cellular modem is not able to communicate serially with the XBee MCU. All communication with the cellular modem must be performed by the user via the USB port.

Configure the data pins

Set [P1 \(DIO11/PWM1 Configuration\)](#) to **7** to configure pins 7 and 8 for USB direct mode.

Enable USB direct mode

If you want to externally control the VBUS pin, set [P0 \(DIO10/PWM0 Configuration\)](#) to **6**.

Apply a logic high signal to DIO10/PWM0 (pin 6) to enable USB or a logic low signal to disable USB.

Note Although that pin is 5 V tolerant on this device, it operates with the same 3.3 V logic as the other XBee device pins. For compatibility with other XBee devices we recommend driving the line with no more than 3.3 V. Moreover, driving the pin at 5 V will cause input leakage current to increase to 3.3 μ A typical.

If you want to enable USB via software, set [DO \(Device Options\)](#) bit 2. Keep in mind that if **P0** is set to **6**, it overrides the behavior of **DO** bit 2.

You must reset the device to enable or disable USB direct mode.

While in USB direct mode, [AI \(Association Indication\)](#) returns 0x2B.

Enable the VBUS option

The VBUS option only applies when the USB direct mode is enabled.

1. Set **P0 (DIO10/PWM0 Configuration)** to **6** to enable USB VBUS. When **P0** is set to **6** the input signal to DIO10/PWM0 (pin 6) propagates to the VBUS line of the cellular component.
2. You can also set **DO (Device Options)** bit 2 to manually control the VBUS signal. When **DO** bit 2 is set, the VBUS signal propagated to the VBUS line of the cellular component is on; when bit 2 is not set the VBUS signal is off.

When **P0** is set to **6**, it overrides the behavior of **DO** bit 2.

Command mode

Command mode is a state in which the firmware interprets incoming characters as commands. It allows you to modify the device's configuration using parameters you can set using AT commands. When you want to read or set any parameter of the device when operating in Transparent mode, you have to enter Command mode and send an AT command. Every AT command starts with the letters **AT** followed by the two characters that identify the command and then by some optional configuration values.

The three operating modes are controlled by the **AP (API Enable)** setting, but Command mode is always available as a mode the XBee Smart Modem can enter while configured for any of the operating modes.

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

Enter Command mode

To get a device to switch into this mode, you must issue the following sequence: **+++** within one second. There must be at least one second preceding and following the **+++** sequence. Both the command character (**CC**) and the silence before and after the sequence (**GT**) are configurable. When the device sees a full second of silence in the data stream (the guard time, **GT**) followed by the string **+++** (without Enter or Return) and another full second of silence, it knows to stop sending data and start accepting commands locally.

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

When the device is in Command mode, it listens for user input and is able to receive AT commands on the UART. If **CT** time (default is 10 seconds) passes without any user input, the device drops out of Command mode and returns to the previous operating mode (Transparent, Bypass, API, Python, and so forth).

You can customize the command character, the guard times and the timeout in the device's configuration settings. For more information, see **CC (Command Sequence Character)**, **CT (Command Mode Timeout)** and **GT (Guard Times)**.

Troubleshooting

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

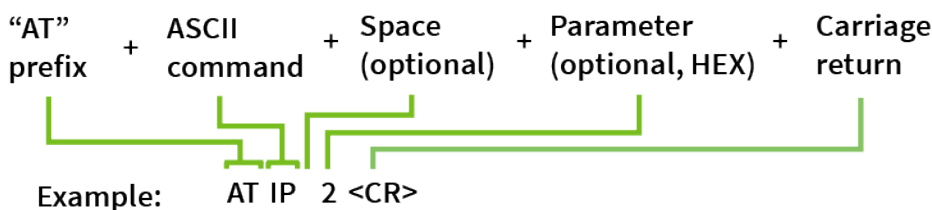
There are two alternative ways to enter Command mode:

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

Both of these methods temporarily set the device's baud rate to 9600 and return an **OK** on the UART to indicate that Command mode is active. When Command mode exits, the device returns to normal operation at the baud rate the **BD** parameter is set to.

Send AT commands

Once the device enters Command mode, use the syntax in the following figure to send AT commands. Every AT command starts with the letters **AT**, which stands for "attention." The **AT** is followed by two characters that indicate which command is being issued, then by some optional configuration values. To read a parameter value stored in the device's register, omit the parameter field.



The preceding example changes the IP protocol to SMS.

Multiple AT commands

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

Parameter format

Refer to the list of AT commands for the format of individual AT command parameters. Valid formats for hexadecimal values include with or without a leading **0x** for example **FFFF** or **0xFFFF**.

Response to AT commands

When reading parameters, the device returns the current parameter value instead of an **OK** message.

Apply command changes

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

1. Send the **AC** (Apply Changes) command.
- or:
2. Exit Command mode.

Make command changes permanent

Issue a **WR** command to save the changes. WR writes parameter values to non-volatile memory so that parameter modifications persist through subsequent resets.

Exit Command mode

1. Send **CN (Exit Command mode)** followed by a carriage return.
or:
2. If the device does not receive any valid AT commands within the time specified by **CT (Command Mode Timeout)**, it returns to the mode that the device was last in.

Sleep modes

About sleep modes	84
Normal mode	84
Pin sleep mode	84
Cyclic sleep mode	84
Cyclic sleep with pin wake up mode	84
Airplane mode	84
SPI mode and sleep pin functionality	84
The sleep timer	85
MicroPython sleep behavior	85

About sleep modes

A number of low-power modes exist to enable devices to operate for extended periods of time on battery power. Use [SM \(Sleep Mode\)](#) to enable these sleep modes.

Normal mode

Set **SM** to 0 to enter Normal mode.

Normal mode is the default sleep mode. If a device is in this mode, it does not sleep and is always awake.

Devices in Normal mode are typically mains powered.

Pin sleep mode

Set **SM** to 1 to enter pin sleep mode.

Pin sleep allows the device to sleep and wake according to the state of the SLEEP_RQ pin (SLEEP_RQ).

When you assert SLEEP_RQ (high), the device finishes any transmit or receive operations, closes any active connection, and enters a low-power state.

When you de-assert SLEEP_RQ (low), the device wakes from pin sleep.

Cyclic sleep mode

Set **SM** to 4 to enter Cyclic sleep mode.

Cyclic sleep allows the device to sleep for a specific time and wake for a short time to poll.

If you use the **D7** command to enable hardware flow control, the $\overline{\text{CTS}}$ pin asserts (low) when the device wakes and can receive serial data, and de-asserts (high) when the device sleeps.

Cyclic sleep with pin wake up mode

Set **SM** to 5 to enter Cyclic sleep with pin wake up mode.

This mode is a slight variation on Cyclic sleep mode (**SM** = 4) that allows you to wake a device prematurely by de-asserting the SLEEP_RQ pin (SLEEP_RQ).

In this mode, you can wake the device after the sleep period expires, or if a high-to-low transition occurs on the SLEEP_RQ pin.

Airplane mode

While not technically a sleep mode, airplane mode is another way of saving power. When set, the cellular component of the XBee Smart Modem is fully turned off and no access to the cellular network is performed or possible. Use [AM \(Airplane Mode\)](#) to configure this mode.

SPI mode and sleep pin functionality

SLEEP_RQ/ DIO8 is configured as a peripheral by default and is used for pin sleep to wake the XBee Smart Modem and put it to sleep. This applies regardless of if the serial interface is UART or SPI.

However, if SLEEP_RQ is not configured as a peripheral and SPI_SSEL is configured as a peripheral, then pin sleep is controlled by SPI_SSEL rather than by SLEEP_RQ. Asserting SPI_SSEL by driving it low

wakes the XBee Smart Modem, or keeps it awake. De-asserting $\overline{\text{SPI_SSEL}}$ by driving it high puts the device to sleep.

If neither pin is configured as a peripheral, then the device stays awake, being unable to sleep when **SM (Sleep Mode)** is **1**.

DIO8/SLEEP_RQ configured as peripheral (D8 = 1)?	DIO3/ $\overline{\text{SPI_SSEL}}$ configured as peripheral (D3 = 1)?	Pin sleep controlled by...
Yes	Yes	DIO8/SLEEP_RQ
Yes	No	DIO8/SLEEP_RQ
No	Yes	$\overline{\text{DIO3/SPI_SSEL}}$
No	No	Neither (pin sleep does not work)

Advantage of using $\overline{\text{SPI_SSEL}}$ to control sleep:

- One less physical pin connection is required to implement pin sleep. This makes DIO8/SLEEP_RQ available for another purpose.

Disadvantages of using $\overline{\text{SPI_SSEL}}$ to control sleep:

- The XBee Smart Modem is put to sleep whenever the SPI master negates $\overline{\text{SPI_SSEL}}$, even if that was not the intent.
- The XBee Smart Modem begins entering sleep as soon as the control pin is asserted (brought high). Immediately de-asserting the control pin (bringing it low) only has the effect of preventing the microcontroller from entering low-power mode before waking up the device - all other sleep preparations (such as closing sockets) continue as in typical sleep operation. This can take several seconds, and this added time in the case of an unintended sleep request may not be acceptable.

The sleep timer

If the device receives serial or RF data in Cyclic sleep mode and Cyclic sleep with pin wake up modes (**SM** = 4 or **SM** = 5), it starts a sleep timer (time until sleep).

- Use **ST (Wake Time)** to set the duration of the timer.
- When the sleep timer expires the device returns to sleep.

MicroPython sleep behavior

When the XBee Smart Modem enters deep sleep mode, any MicroPython code currently executing is suspended until the device comes out of sleep. When the XBee Smart Modem comes out of sleep mode, MicroPython execution continues where it left off.

Upon entering deep sleep mode, the XBee Smart Modem closes any active UDP connections and turns off the cellular component. As a result, any sockets that were opened in MicroPython prior to sleep report as no longer being connected. This behavior appears the same as a typical socket disconnection event will:

- **socket.send** raises **OSError: ENOTCONN**
- **socket.sendto** raises **OSError: ENOTCONN**
- **socket.recv** returns the empty string, the traditional end-of-file return value
- **socket.recvfrom** returns an empty message, for example:
(b'', (<address from connect()>, <port from connect()>))

The underlying UDP socket resources have been released at this point.

Serial communication

- Serial interface 88
- Serial data 88
- UART data flow 88
- Serial buffers 89
- CTS flow control 89
- RTS flow control 89
- Enable UART or SPI ports 89

Serial interface

The XBee Smart Modem interfaces to a host device through a serial port. The device can communicate through its serial port with:

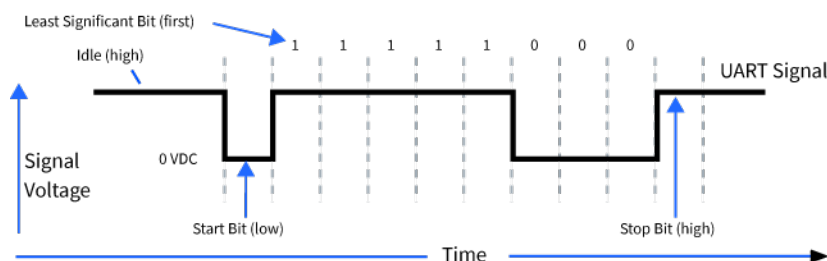
- Through logic and voltage compatible universal asynchronous receiver/transmitter (UART).
- Through a level translator to any serial device, for example, through an RS-232 or USB interface board.
- Through a serial peripheral interface (SPI) port.

Serial data

A device sends data to the XBee Smart Modem's UART through pin 3 DIN as an asynchronous serial signal. When the device is not transmitting data, the signals should idle high.

For serial communication to occur, you must configure the UART of both devices (the microcontroller and the XBee Smart Modem) with compatible settings for the baud rate, parity, start bits, stop bits, and data bits.

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

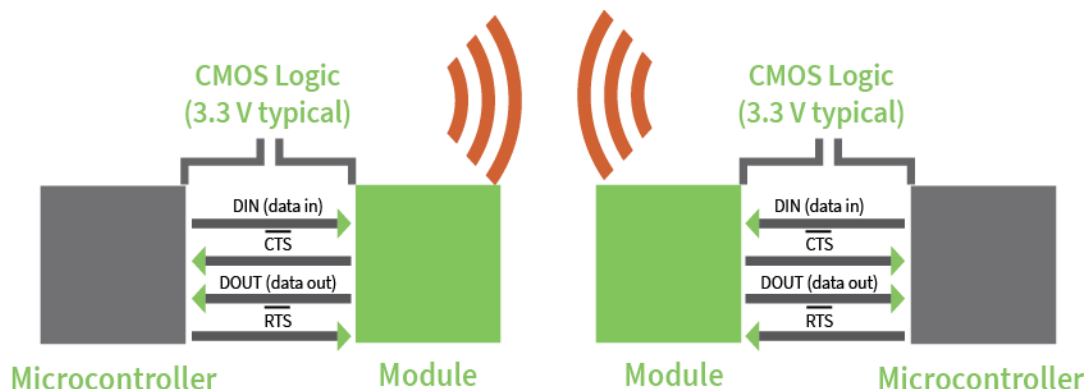


You can configure the UART baud rate, parity, and stop bits settings on the device with the **BD**, **NB**, and **SB** commands respectively. For more information, see [Serial interfacing commands](#).

In the rare case that a device has been configured with the UART disabled, you can recover the device to UART operation by holding DIN low at reset time. DIN forces a default configuration on the UART at 9600 baud and it brings the device up in Command mode on the UART port. You can then send the appropriate commands to the device to configure it for UART operation. If those parameters are written, the device comes up with the UART enabled on the next reset.

UART data flow

Devices that have a UART interface connect directly to the pins of the XBee Smart Modem as shown in the following figure. The figure shows system data flow in a UART-interfaced environment. Low-asserted signals have a horizontal line over the signal name.



Serial buffers

The XBee Smart Modem maintains internal buffers to collect serial and RF data that it receives. The serial receive buffer collects incoming serial characters and holds them until the device can process them. The serial transmit buffer collects the data it receives via the RF link until it transmits that data out the serial or SPI port.

CTS flow control

We strongly encourage you to use flow control with the XBee Smart Modem to prevent buffer overruns.

CTS flow control is enabled by default; you can disable it with [D7 \(DIO7/CTS\)](#). When the serial receive buffer fills with the number of bytes specified by [FT \(Flow Control Threshold\)](#), the device de-asserts CTS (sets it high) to signal the host device to stop sending serial data. The device re-asserts CTS when less than FT-32 bytes are in the UART receive buffer.

Note Serial flow control is not possible when using the SPI port.

RTS flow control

If you set [D6 \(DIO6/RTS\)](#) to enable RTS flow control, the device does not send data in the serial transmit buffer out the DOUT pin as long as RTS is de-asserted (set high). Do not de-assert RTS for long periods of time or the serial transmit buffer will fill.

Enable UART or SPI ports

To enable the UART port, configure DIN and DOUT (**P3** and **P4** parameters) as peripherals. To enable the SPI port, enable SPI_MISO, SPI_MOSI, SPI_SSEL, and SPI_CLK (**P5** through **P9**) as peripherals. If you enable both ports then output goes to the UART until the first input on SPI.

When both the UART and SPI ports are enabled on power-up, all serial data goes out the UART. As soon as input occurs on either port, that port is selected as the active port and no input or output is allowed on the other port until the next device reset.

If you change the configuration so that only one port is configured, then that port is the only one enabled or used. If the parameters are written with only one port enabled, then the port that is not enabled is not used even temporarily after the next reset.

If both ports are disabled on reset, the device uses the UART in spite of the wrong configuration so that at least one serial port is operational.

SPI operation

SPI communications	92
Full duplex operation	93
Low power operation	94
Select the SPI port	94
Force UART operation	95
Data format	95

SPI communications

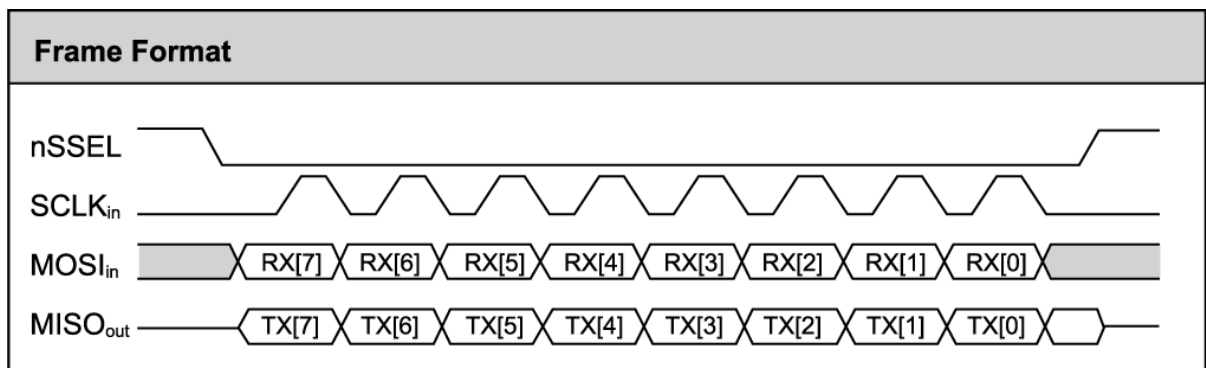
The XBee Smart Modem supports SPI communications in slave mode. Slave mode receives the clock signal and data from the master and returns data to the master. The following table shows the signals that the SPI port uses on the device.

Signal	Function
SPI_MOSI (Master Out, Slave In)	Inputs serial data from the master
SPI_MISO (Master In, Slave Out)	Outputs serial data to the master
SPI_SCLK (Serial Clock)	Clocks data transfers on MOSI and MISO
SPI_SSEL (Slave Select)	Enables serial communication with the slave
SPI_ATTN (Attention)	Alerts the master that slave has data queued to send. The XBee Smart Modem asserts this pin as soon as data is available to send to the SPI master and it remains asserted until the SPI master has clocked out all available data.

In this mode:

- SPI clock rates up to 6 MHz are possible.
- Data is most significant bit (MSB) first; bit 7 is the first bit of a byte sent over the interface.
- Frame Format mode 0 is used. This means CPOL= 0 (idle clock is low) and CPHA = 0 (data is sampled on the clock's leading edge).
- The SPI port only supports API Mode (**AP = 1**).

The following diagram shows the frame format mode 0 for SPI communications.



SPI mode is chip to chip communication. We do not supply a SPI communication option on the device development evaluation boards.

Full duplex operation

The specification for SPI includes the four signals SPI_MISO, SPI_MOSI, SPI_CLK, and SPI_SSEL. Using these four signals, the SPI master cannot know when the slave needs to send and the SPI slave cannot transmit unless enabled by the master. For this reason, the SPI_ATTN signal is available in the design. This allows the SPI slave to alert the SPI master that it has data to send. In turn, the SPI master is expected to assert SPI_SSEL and start SPI_CLK, unless these signals are already asserted and active respectively. This, in turn, allows the XBee Smart Modem SPI slave to send data to the master.

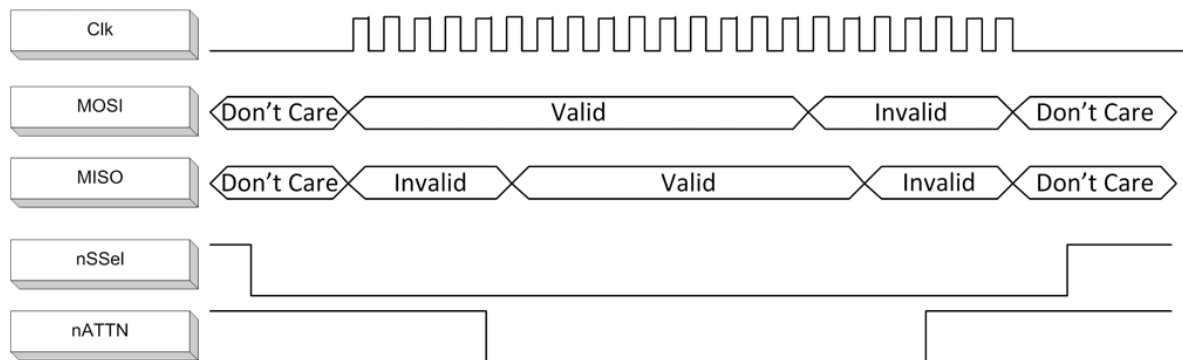
SPI data is latched by the master and slave using the SPI_CLK signal. When data is being transferred the MISO and MOSI signals change between each clock. If data is not available then these signals will not change and will be either 0 or 1. This results in receiving either a repetitive 0 or 0xFF. The means of determining whether or not received data is valid is by packetizing the data with API packets, without escaping. Valid data to and from the XBee Smart Modem is delimited by 0x7E, a length, the payload, and finally a checksum byte. Everything else in both directions should be ignored. The bytes received between frames will be either 0xff or 0x00. This allows the SPI master to scan for a 0x7E delimiter between frames.

SPI allows for valid data from the slave to begin before, at the same time, or after valid data begins from the master. When the master is sending data to the slave and the slave has valid data to send in the middle of receiving data from the master, it allows a true full duplex operation where data is valid in both directions for a period of time. During this time, the master and slave must simultaneously transmit valid data at the clock speed so that no invalid bytes appear within an API frame, causing the whole frame to be discarded.

An example follows to more fully illustrate the SPI interface during the time valid data is being sent in both directions. First, the master asserts SPI_SSEL and starts SPI_CLK to send a frame to the slave.

Initially, the slave does not have valid data to send the master. However, while it is still receiving data from the master, it has its own data to send. Therefore, it asserts SPI_ATTN low. Seeing that SPI_SSEL is already asserted and that SPI_CLK is active, it immediately begins sending valid data, even while it is receiving valid data from the master. In this example, the master finishes its valid data before the slave does. The master will have two indications of valid data: The SPI_ATTN line is asserted and the API frame length is not yet expired. For both of these reasons, the master should keep SPI_SSEL asserted and should keep SPI_CLK toggling in order to receive the end of the frame from the slave, even though these signals were originally turned on by the master to send data. During the time that the SPI master is sending invalid data to the SPI slave, it is important no 0x7E is included in that invalid data because that would trigger the SPI slave to start receiving another valid frame.

The following figure illustrates the SPI interface while valid data is being sent in both directions.



Low power operation

Sleep modes generally work the same on SPI as they do on UART. However, due to the addition of SPI mode, there is an option of another sleep pin, as described below.

By default, Digi configures DIO8 (SLEEP_REQUEST) as a peripheral and during pin sleep it wakes the device and puts it to sleep. This applies to both the UART and SPI serial interfaces.

If SLEEP_REQUEST is not configured as a peripheral and SPI_SSEL is configured as a peripheral, then pin sleep is controlled by SPI_SSEL rather than by SLEEP_REQUEST. Asserting SPI_SSEL (pin 17) by driving it low either wakes the device or keeps it awake. Negating SPI_SSEL by driving it high puts the device to sleep.

Using SPI_SSEL to control sleep and to indicate that the SPI master has selected a particular slave device has the advantage of requiring one less physical pin connection to implement pin sleep on SPI. It has the disadvantage of putting the device to sleep whenever the SPI master negates SPI_SSEL (meaning time is lost waiting for the device to wake), even if that was not the intent.

If the user has full control of SPI_SSEL so that it can control pin sleep, whether or not data needs to be transmitted, then sharing the pin may be a good option in order to make the SLEEP_REQUEST pin available for another purpose.

If the device is one of multiple slaves on the SPI, then the device sleeps while the SPI master talks to the other slave, but this is acceptable in most cases.

If you do not configure either pin as a peripheral, then the device stays awake, being unable to sleep in **SM1** mode.

Select the SPI port

To force SPI mode, hold DOUT/DIO13 pin 2 low while resetting the device until SPI_ATTEN asserts. This causes the device to disable the UART and go straight into SPI communication mode. Once configuration is complete, the device queues a modem status frame to the SPI port, which causes the SPI_ATTEN line to assert. The host can use this to determine that the SPI port is configured properly. This method forces the configuration to provide full SPI support for the following parameters:

- **D1** (This parameter will only be changed if it is at a default of zero when the method is invoked.)
- **D2**
- **D3**
- **D4**
- **P2**

As long as the host does not issue a **WR** command, these configuration values revert to previous values after a power-on reset. If the host issues a **WR** command while in SPI mode, these same parameters are written to flash. After a reset, parameters that were forced and then written to flash become the mode of operation.

If the UART is disabled and the SPI is enabled in the written configuration, then the device comes up in SPI mode without forcing it by holding DOUT low. If both the UART and the SPI are enabled at the time of reset, then output goes to the UART until the host sends the first input. If that first input comes on the SPI port, then all subsequent output goes to the SPI port and the UART is disabled. If the first input comes on the UART, then all subsequent output goes to the UART and the SPI is disabled.

Once you select a serial port (UART or SPI), all subsequent output goes to that port, even if you apply a new configuration. The only way to switch the selected serial port is to reset the device. On surface-mount devices, forcing DOUT low at the time of reset has no effect. To use SPI mode on the SMT modules, assert the SPI_SSEL (pin 17) low after reset and before any UART data is input.

When the master asserts the slave select (SPI_SSEL) signal, SPI transmit data is driven to the output pin SPI_MISO, and SPI data is received from the input pin SPI_MOSI. The SPI_SSEL pin has to be asserted to enable the transmit serializer to drive data to the output signal SPI_MISO. A rising edge on SPI_SSEL causes the SPI_MISO line to be tri-stated such that another slave device can drive it, if so desired.

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

Force UART operation

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

Data format

SPI only operates in API mode 1. The XBee Smart Modem does not support Transparent mode or API mode 2 (which escapes control characters). This means that the AP configuration only applies to the UART, and the device ignores it while using SPI. The reason for this operation choice is that SPI is full duplex. If data flows in one direction, it flows in the other. Since it is not always possible to have valid data flowing in both directions at the same time, the receiver must have a way to parse out the valid data and to ignore the invalid data.

Officially, the invalid data is undefined for SPI. The only requirement is that the start of frame byte (0x7E) cannot be included in the invalid data as this would cause the receiver to begin parsing a new frame. But, in reality the XBee Smart Modem sends 0xFF for invalid characters and the S8 and S6 products may send all ones or all zeros.

AT commands

Special commands	97
Cellular commands	98
Network commands	100
Addressing commands	102
Serial interfacing commands	105
I/O settings commands	107
I/O sampling commands	115
Sleep commands	116
Command mode options	117
MicroPython commands	117
Firmware version/information commands	119
Execution commands	122

Special commands

The following commands are special commands.

AC (Apply Changes)

Immediately applies new settings without exiting Command mode.

Applying changes means that the device re-initializes based on changes made to its parameter values. Once changes are applied, the device immediately operates according to the new parameter values.

This behavior is in contrast to issuing the **WR** (Write) command. The **WR** command saves parameter values to non-volatile memory, but the device still operates according to previously saved values until the device is rebooted or you issue the **CN** (Exit AT Command Mode) or **AC** commands.

Parameter range

N/A

Default

N/A

FR (Force Reset)

Resets the device. The device responds immediately with an **OK** and performs a reset 100 ms later.

If you issue **FR** while the device is in Command Mode, the reset effectively exits Command mode.

Note We recommend entering Airplane mode before resetting or rebooting the device to allow the cellular module to detach from the network.

Parameter range

N/A

Default

N/A

RE command

Restore device parameters to factory defaults.

The **RE** command does not write restored values to non-volatile (persistent) memory. Issue the **WR** (Write) command after issuing the **RE** command to save restored parameter values to non-volatile memory.

Parameter range

N/A

Default

N/A

WR command

Writes parameter values to non-volatile memory so that parameter modifications persist through subsequent resets.

Note Once you issue a **WR** command, do not send any additional characters to the device until after you receive the **OK** response.

Parameter range

N/A

Default

N/A

Cellular commands

The following AT commands are cellular configuration and data commands.

PH (Phone Number)

Reads the SIM card phone number. If **PH** is blank, the XBee Smart Modem is not registered to the network.

Parameter range

N/A

Default

Set by the cellular carrier via the SIM card

S# (ICCID)

Reads the Integrated Circuit Card Identifier (ICCID) of the inserted SIM.

Parameter range

N/A

Default

Set by the SIM card

IM (IMEI)

Reads the device's International Mobile Equipment Identity (IMEI).

Parameter range

N/A

Default

Set in the factory

MN (Operator)

Reads the network operator on which the device is registered.

Parameter range

N/A

Default

N/A

MV (Modem Firmware Version)

Read the firmware version string for cellular component communications. See the related [VR \(Firmware Version\)](#) command.

Parameter range

N/A

Default

Set in the currently loaded firmware

DB (Cellular Signal Strength)

Reads the absolute value of the current signal strength to the cell tower in dB. If **DB** is blank, the XBee Smart Modem has not received a signal strength from the cellular component.

Parameter range

0x71 - 0x33 (-113 dBm to -51 dBm) [read-only]

Default

N/A

AN (Access Point Name)

Specifies the packet data network that the modem uses for Internet connectivity. This information is provided by your cellular network operator. After you set this value, applying changes with [AC \(Apply Changes\)](#) or [CN \(Exit Command mode\)](#) triggers a network reset.

See [Network connection issues](#) if the XBee Smart Modem is not joining the network.

Parameter range

1 - 100 ASCII characters

Default

-

AM (Airplane Mode)

When set, the cellular component of the XBee Smart Modem is fully turned off and no access to the cellular network is performed or possible.

Parameter range

0 - 1

0 = Normal operation

1 = Airplane mode

Default

0

Network commands

The following commands are network commands.

IP (IP Protocol)

Sets or displays the IP protocol used for client and server socket connections in IP socket mode.

Parameter range

0 - 4

Value	Description
0x00	UDP
0x01	TCP
0x02	Reserved
0x03	Reserved
0x04	Reserved

Default

0x01

TL (SSL/TLS Protocol Version)

Sets the SSL/TLS protocol version used for the SSL socket. If you change the **TL** value, it does not affect any currently open sockets. The value only applies to subsequently opened sockets.

Note Due to known vulnerabilities in prior protocol versions, we strongly recommend that you use the latest TLS version whenever possible.

Range

Value	Description
0x00	SSL v3
0x01	TLS v1.0
0x02	TLS v1.1
0x03	TLS v1.2

Default

0x03

TM (IP Client Connection Timeout)

The IP client connection timeout. If there is no activity for this timeout then the connection is closed. If **TM** is **0**, the connection is closed immediately after the device sends data.

If you change the **TM** value while in Transparent Mode, the current connection is immediately closed. Upon the next transmission, the **TM** value applies to the newly created socket.

If you change the **TM** value while in API Mode, the value only applies to subsequently opened sockets.

Parameter range

0 - 0xFFFF [x 100 ms]

Default

0xBB8 (5 minutes)

TS (IP Server Connection Timeout)

The IP server connection timeout. If no activity for this timeout then the connection is closed. When set to **0** the connection is closed immediately after data is sent.]

Parameter Range

10 - 0xFFFF; (x 100 ms)

Default

3000

DO (Device Options)

Enables and disables special features on the XBee Smart Modem.

Enables and disables special features on the XBee Smart Modem according to the following table.

Bit 0 - Remote Manager support

If the XBee Smart Modem cannot establish a connection with Remote Manager , it waits 30 seconds before trying again. On each successive connection failure, the wait time doubles (60 seconds, 120, 240, and so on) up to a maximum of 1 hour. This time resets to 30 seconds once the connection to Remote Manager succeeds or if the device is reset.

After changing this setting, you must:

1. Use [WR command](#) to write all values to flash.
2. Use [FR \(Force Reset\)](#) to reset the device.
3. Wait for the cellular component to be initialized: [AI \(Association Indication\)](#) reaches **0x00**.

Bits 1, 3-7

Reserved

Bit 2 - USB direct enable

Set bit 2 to enable USB direct mode. After setting, use [WR command](#) to write all values to flash and use [FR \(Force Reset\)](#) to reset the device.

Note Setting [P0 \(DIO10/PWM0 Configuration\)](#) to **6** overrides setting **DO** bit 2.

Range

0x00 - 0x07

The supported states of **DO** are:

Value	Remote Manager	USB direct
0x00	Disabled	Disabled
0x01	Enabled	Disabled
0x04	Disabled	Enabled
0x05	Enabled	Enabled

Default

0x01

EQ (Device Cloud FQDN)

Sets or display the fully qualified domain name of the Remote Manager server.

Range

From 0 through 63 ASCII characters.

Default

my.devicecloud.com

Addressing commands

The following AT commands are addressing commands.

SH (Serial Number High)

The upper digits of the unique International Mobile Equipment Identity (IMEI) assigned to this device.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

N/A

SL (Serial Number Low)

The lower digits of the unique International Mobile Equipment Identity (IMEI) assigned to this device.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

N/A

MY (Module IP Address)

Reads the device's IP address. This command is read-only because the IP address is assigned by the mobile network.

In API mode, the address is represented as the binary four byte big-endian numeric value representing the IPv4 address.

In Transparent or Command mode, the address is represented as a dotted-quad string notation.

Parameter range

0- 15 IPv4 characters

Default

0.0.0.0

P# (Destination Phone Number)

Sets or displays the destination phone number used for SMS when **IP (IP Protocol) = 2**. Phone numbers must be fully numeric, 7 to 20 ASCII digits, for example: 8889991234.

P# allows international numbers with or without the **+** prefix. If you omit **+** and are dialing internationally, you need to include the proper International Dialing Prefix for your calling region, for example, 011 for the United States.

Range

7 - 20 ASCII digits including an optional **+** prefix

Default

N/A

N1 (DNS Address)

Displays the IPv4 address of the primary domain name server.

Parameter Range

Read-only

Default

0.0.0.0 (waiting on cellular connection)

N2 (DNS Address)

Displays the IPv4 address of the secondary domain name server.

Parameter Range

Read-only

Default

0.0.0.0 (waiting on cellular connection)

DL (Destination Address)

The destination IPv4 address or fully qualified domain name.

To set the destination address to an IP address, the value must be a dotted quad, for example **XXX.XXX.XXX.XXX**.

To set the destination address to a domain name, the value must be a legal Internet host name, for example **remotemanager.digi.com**

Parameter range

0 - 128 ASCII characters

Default

0.0.0.0

OD (Operating Destination Address)

Read the destination IPv4 address currently in use by Transparent mode. The value is **0.0.0.0** if no Transparent IP connection is active.

In API mode, the address is represented as the binary four byte big-endian numeric value representing the IPv4 address.

In Transparent or Command mode, the address is represented as a dotted-quad string notation.

Parameter range

-

Default

0.0.0.0

DE (Destination Port)

Sets or displays the destination IP port number.

Parameter range

0x0 - 0xFFFF

Default

0x2616

C0 (Source Port)

Set or get the port number used to provide the serial communication service. Data received by this port on the network is transmitted on the XBee Smart Modem's serial port.

As long as a network connection is established to this port (for TCP) data received on the serial port is transmitted on the established network connection.

[IP \(IP Protocol\)](#) sets the protocol used when UART is in Transparent or API mode.

For more information on using incoming connections, see [Socket behavior](#).

Parameter range

0 - 0xFFFF

Value	Description
0	Disabled
Non-0	Enabled on that port

Default

0

LA (Lookup IP Address of FQDN)

Performs a DNS lookup of the given fully qualified domain name (FQDN) and outputs its IP address.

When you issue the command in API mode, the IP address is formatted in binary four byte big-endian numeric value. In all other cases (for example, Command mode) the format is dotted decimal notation.

Range

Valid FQDN

Default

-

Serial interfacing commands

The following AT commands are serial interfacing commands.

BD (Baud Rate)

Sets or displays the serial interface baud rate for communication between the device's serial port and the host.

Modified interface baud rates do not take effect until the XBee Smart Modem exits Command mode or you issue [AC \(Apply Changes\)](#). The baud rate resets to default unless you save it with [WR command](#) or by clicking the **Write module settings** button in XCTU.

Parameter range

Standard baud rates: 0x1 - 0x8

Non-standard baud rates: 0x5B9 to 0x3D090 (250,000 b/s)

Parameter	Description
0x0	1200 b/s
0x1	2400 b/s
0x2	4800 b/s
0x3	9600 b/s
0x4	19200 b/s
0x5	38400 b/s
0x6	57600 b/s
0x7	115200 b/s
0x8	230400 b/s

Default

0x3 (9600 b/s)

NB (Parity)

Set or read the serial parity settings for UART communications.

Parameter range

0x00 - 0x02

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

Default

0x00

SB (Stop Bits)

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

Parameter range

0 - 1

Parameter	Configuration
0	One stop bit
1	Two stop bits

Default

0

RO (Packetization Timeout)

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

RF transmission also starts after 100 bytes (maximum packet size) are received in the **DI** buffer.

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

Set to **FF** for realtime typing by humans. Also, see [TD \(Text Delimiter\)](#).

Parameter range

0 - 0xFF (x character times)

Default

3

TD (Text Delimiter)

The ASCII character used as a text delimiter for Transparent mode. When you select a character, information received over the serial port in Transparent mode is not transmitted until that character is received. To use a carriage return, set to 0xD. Set to zero to disable text delimiter checking.

Parameter range

0 - 0xFF

Default

0x0

FT (Flow Control Threshold)

Set or display the flow control threshold.

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

Parameter range

0x9D - 0x82D

Default

0x681

AP (API Enable)

The API mode setting. The device can format the RF packets it receives into API frames and send them out the UART. When API is enabled the UART data must be formatted as API frames because Transparent mode is disabled. See [Modes](#) for more information.

Parameter range

0x00 - 0x05

Parameter	Description
0x00	API disabled (operate in Transparent mode)
0x01	API enabled
0x02	API enabled (with escaped control characters)
0x03	N/A
0x04	MicroPython REPL
0x05	Bypass mode

Default

0

I/O settings commands

The following AT commands are I/O settings commands.

D0 (DIO0/AD0)

Sets or displays the DIO0/AD0 configuration (pin 20).

Parameter range

0, 2 - 5

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

Default

0

D1 (DIO1/AD1)

Sets or displays the DIO1/AD1 configuration (pin 19).

Parameter range

0 - 6

Parameter	Description
0	Disabled
1	SPI_ATTN
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high
6	I2C SCL

Default

0

D2 (DIO2/AD2)

Sets or displays the DIO2/AD2 configuration (pin 18).

Parameter range

0 - 5

	Description
0	Disabled
1	SPI_CLK
2	Analog input
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

0

D3 (DIO3/AD3)

Sets or displays the DIO3/AD3 configuration (pin 17).

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	SPI_SSEL
2	Analog input
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

0

D4 (DIO4)**Parameter range****Default**

0

D5 (DIO5/ASSOCIATED_INDICATOR)

Sets or displays the DIO5/ASSOCIATED_INDICATOR configuration (pin 15).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Associated LED
2	N/A
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

D6 (DIO6/RTS)

Sets or displays the DIO6/ $\overline{\text{RTS}}$ configuration (pin 16).

Parameter range

0, 1, 3 - 5

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

Default

0

D7 (DIO7/CTS)

Sets or displays the DIO7/ $\overline{\text{CTS}}$ configuration (pin 12).

Parameter range

0, 1, 3 - 5

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

Default

0x1

D8 (DIO8/SLEEP_REQUEST)Sets or displays the DIO8/ $\overline{\text{DTR}}$ /SLP_RQ configuration (pin 9).**Parameter range**

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SLEEP_REQUEST input
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

Sets or displays the DIO9/ON_ $\overline{\text{SLEEP}}$ configuration (pin 13).**Parameter range**

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	ON/ $\overline{\text{SLEEP}}$ output
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

P0 (DIO10/PWM0 Configuration)

Sets or displays the PWM/DIO10 configuration (pin 6).

This command enables the option of translating incoming data to a PWM so that the output can be translated back into analog form.

Parameter range

0 - 6

Parameter	Description
0	Disabled
1	RSSI PWM0 output
2	PWM0 output
3	Digital input
4	Digital output, low
5	Digital output, high
6	USB VBUS

Default

0

P1 (DIO11/PWM1 Configuration)

Sets or displays the DIO11 configuration (pin 7).

Parameter range

0, 1, 3 - 7

Parameter	Description
0	Disabled
1	Fan enable. Output is low when the XBee Smart Modem is sleeping, turning an attached fan off when the cellular component is in a power saving mode, and also during Airplane Mode
3	Digital input
4	Digital output, default low
5	Digital output, default high
6	I2C SDA
7	USB direct

Default

0

P2 (DIO12 Configuration)

Sets or displays the DIO12 configuration (pin 4).

Parameter range

0, 1, 3 - 5

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

Default

0

P3 (DIO13/DOUT)

Sets or displays the DIO13/DOUT configuration (pin 17).

Parameter range

0, 1

Parameter	Description
0	Disabled
1	UART DOUT enabled

Default

1

P4 (DIO14/DIN)

Sets or displays the DIO14/DIN configuration (pin 3).

Parameter range

0 - 1

Parameter	Description
0	Disabled
1	UART DIN enabled

Default

1

PD (Pull Direction)

The resistor pull direction bit field (**1** = pull-up, **0** = pull-down) for corresponding I/O lines that are set by [PR \(Pull-up/down Resistor Enable\)](#).

If the bit is not set in **PR**, the device uses **PD**.

Note Resistors are not applied to disabled lines.

See [PR \(Pull-up/down Resistor Enable\)](#) for bit mappings, which are the same.

Parameter range

0x0 – 0x7FFF

Default

0 – 0x7FFF

PR (Pull-up/down Resistor Enable)

Sets or displays the bit field that configures the internal resistor status for the digital input lines. Internal pull-up/down resistors are not available for digital output pins, analog input pins, or for disabled pins.

Use the **PD** command to specify whether the resistor is pull-up or pull-down.

- If you set a **PR** bit to 1, it enables the pull-up/down resistor
- If you set a **PR** bit to 0, it specifies no internal pull-up/down resistor.

The following table defines the bit-field map for both the **PR** and **PD** commands.

Bit	I/O line	Module pin
0	DIO4	pin 11
1	DIO3/AD3	pin 17
2	DIO2/AD2	pin 18
3	DIO1/AD1	pin 19
4	DIO0/AD0	pin 20
5	DIO6/ $\overline{\text{RTS}}$	pin 16
6	DIO8/SLEEP_REQUEST	pin 9

Bit	I/O line	Module pin
7	DIO14/DIN	pin 3
8	DIO5/ASSOCIATE	pin 15
9	DIO9/On/SLEEP	pin 13
10	DIO12	pin 4
11	DIO10	pin 6
12	DIO11	pin 7
13	DIO7/CTS	pin 12
14	DIO13/DOUT	pin 17

Parameter range

0 - 0x7FFF (bit field)

Default

0x7FFF

M0 (PWM0 Duty Cycle)

Sets the duty cycle of PWM0 (pin 6) for **P0** = **2**, where a value of 0x200 is a 50% duty cycle.

Before setting the line as an output:

1. Enable PWM0 output ([P0 \(DIO10/PWM0 Configuration\)](#) = **2**).
2. Apply the settings (use [CN command](#) or [AC \(Apply Changes\)](#)).

The PWM period is 42.62 μ s and there are 0x03FF (1023 decimal) steps within this period. When **M0** = **0** (0% PWM), **0x01FF** (50% PWM), **0x03FF** (100% PWM), and so forth.

Parameter range

0 - 0x3FF

Default

0

I/O sampling commands

The following AT commands configure I/O sampling parameters.

TP (Temperature)

Displays the temperature of the XBee Smart Modem in degrees Celsius. The temperature value is displayed in 8-bit two's complement format. For example, **0x1A** = 26 °C, and **0xF6** = -10 °C.

Parameter range

0 - 0xFF which indicates degrees Celsius displayed in 8-bit two's complement format.

Default

N/A

Sleep commands

The following AT commands are sleep commands.

SM (Sleep Mode)

Sets or displays the sleep mode of the device.

The sleep mode determines how the device enters and exits a power saving sleep.

Sleep mode is also affected by the **SO** command, option bit 6. See [Sleep modes](#) for more information about sleep modes.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Normal. In this mode the device never sleeps.
1	Pin Sleep. In this mode the device honors the SLEEP_RQ pin. Set D8 (DIO8/SLEEP_REQUEST) to the sleep request function: 1 .
4	Cyclic Sleep. In this mode the device repeatedly sleeps for the value specified by SP and spends ST time awake.
5	Cyclic Sleep with Pin Wake. In this mode the device acts as in Cyclic Sleep but does not sleep if the SLEEP_RQ pin is inactive, allowing the device to be kept awake or woken by the connected system.

Default

0

SP (Sleep Period)

Sets or displays the time to spend asleep in cyclic sleep modes. In Cyclic sleep mode, the node sleeps with CTS disabled for the sleep time interval, then wakes for the wake time interval.

Parameter range

0x1 - 0x83D600 (x 10 ms)

Default

0x7530 (5 minutes)

ST (Wake Time)

Sets or displays the time to spend awake in cyclic sleep modes.

Parameter range

0x1 - 0x36EE80 (x 1 ms)

Default

0xEA60 (60 seconds)

Command mode options

The following commands are Command mode option commands.

CC (Command Sequence Character)

The character value the device uses to enter Command mode.

The default value (**0x2B**) is the ASCII code for the plus (+) character. You must enter it three times within the guard time to enter Command mode. To enter Command mode, there is also a required period of silence before and after the command sequence characters of the Command mode sequence (**GT + CC + GT**). The period of silence prevents inadvertently entering Command mode.

Parameter range

0 - 0xFF

Default

0x2B (the ASCII plus character: +)

CT (Command Mode Timeout)

Sets or displays the Command mode timeout parameter. If a device does not receive any valid commands within this time period, it returns to Idle mode from Command mode.

Parameter range

2 - 0x1770 (x 100 ms)

Default

0x64 (10 seconds)

GT (Guard Times)

Set the required period of silence before and after the command sequence characters of the Command mode sequence (**GT + CC + GT**). The period of silence prevents inadvertently entering Command mode.

Parameter range

0x2 - 0x6D3 (x 1 ms)

Default

0x3E8 (one second)

MicroPython commands

The following commands relate to using MicroPython on the XBee Smart Modem.

PS (Python Startup)

Sets whether or not the XBee Smart Modem runs the stored Python code at startup.

Range

0 - 1

Parameter	Description
0	Do not run stored Python code at startup.
1	Run stored Python code at startup.

Default

0

PY (MicroPython Command)

Interact with the XBee Smart Modem using MicroPython. **PY** is a command with sub-commands. These sub-commands are arguments to **PY**.

PYC(Code Report)

You can store compiled code in flash using the **Ctrl-F** command from the MicroPython REPL; refer to the [Digi MicroPython Programming Guide](#). The **PYC** sub-command reports details of the stored code. In Command mode, it returns three lines of text, for example:

```
source: 1662 bytes (hash=0xC3B3A813)
bytecode: 619 bytes (hash=0x0900DBCE)
compiled: 2017-05-09T15:49:44
```

The messages are:

- **source**: the size of the source code used to generate the bytecode and its 32-bit hash.
- **bytecode**: the size of bytecode stored in flash and its 32-bit hash. A size of **0** indicates that there is no stored code.
- **compiled**: a compilation timestamp. A timestamp of **2000-01-01T00:00:00** indicates that the clock was not set during compilation.

In API mode, **PYC** returns five 32-bit big-endian values:

- source size
- source hash
- bytecode size
- bytecode hash
- timestamp as seconds since 2000-01-01T00:00:00

PYD (Delete Code)

PYD interrupts any running code, erases any stored code and then does a soft-reboot on the MicroPython subsystem.

PYV (Version Report)

Report the MicroPython version.

PY^ (Interrupt Program)

Sends **KeyboardInterrupt** to MicroPython. This is useful if there is a runaway MicroPython program and you have filled the stdin buffer. You can enter Command mode (**+++**) and send **ATPY^** to interrupt the program.

Default

N/A

Firmware version/information commands

The following AT commands are firmware version/information commands.

VR (Firmware Version)

Reads the firmware version on a device.

Parameter range

0 - 0xFFFF [read-only]

Default

Set in firmware

VL (Verbose Firmware Version)

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

Parameter range

N/A

Default

Set in firmware

HV (Hardware Version)

Read the device's hardware version. Use this command to distinguish between different hardware platforms. The upper byte returns a value that is unique to each device type. The lower byte indicates the hardware revision.

Parameter range

0 - 0xFFFF [read-only]

Default

Set in firmware

AI (Association Indication)

Reads the Association status code to monitor association progress. The following table provides the status codes and their meanings.

Status code	Meaning
0x00	Connected to the Internet.
0x22	Registering to cellular network.
0x23	Connecting to the Internet.
0x24	The cellular component is missing, corrupt, or otherwise in error. The cellular component requires a new firmware image.
0x25	Cellular network registration denied.
0x2A	Airplane mode.
0x2B	USB Direct active.
0x2F	Bypass mode active.
0xFF	Initializing.

Parameter range

0 - 0xFF [read-only]

Default

N/A

DI (Device Cloud Indicator)

Displays the current Remote Manager status for the XBee.

Range

Value	Description
0x00	Connected
0x01	Before connection to the Internet
0x02	Remote Manager connection in progress
0x03	Disconnecting from Remote Manager
0x04	Not configured for Remote Manager

Default

N/A

CI (Protocol/Connection Indication)

Displays information regarding the last IP connection (when the **IP** command = **0** or **1**).

The following table provides the parameter's meaning when **IP** = **0** for UDP connections.

Parameter	Description
0x00	The socket is open.
0x01	Tried to send but could not.
0x02	Invalid parameters (bad IP/host).
0x03	TCP not supported on this cellular component.
0x10	Not registered to the cell network.
0x11	Cellular component not identified yet.
0x12	DNS query lookup failure.
0x20	Bad handle.
0x21	User closed.
0x22	Unknown server - DNS lookup failed.
0x23	Connection lost.
0x24	Unknown.
0xFF	No known status.

The following table provides the parameter's meaning when **IP = 1** or **4** for TCP connections.

Parameter	Description
0x00	The socket is open.
0x01	Tried to send but could not.
0x02	Invalid parameters (bad IP/host).
0x03	TCP not supported on this cellular component.
0x10	Not registered to the cell network.
0x11	Cellular component not identified yet.
0x12	DNS query lookup failure.
0x20	Bad handle.
0x21	User closed.
0x22	No network registration.
0x23	No internet connection.
0x24	No server - timed out on connection.
0x25	Unknown server - DNS lookup failed.
0x26	Connection refused.

Parameter	Description
0x27	Connection lost.
0x28	Unknown.
0xFF	No known status.

Parameter range

0 - 0xFF (read-only)

Default

-

HS (Hardware Series)

Read the device's hardware series number.

Parameter range

N/A

Default

Set in the firmware

CK (Configuration CRC)

Displays the cyclic redundancy check (CRC) of the current AT command configuration settings.

Parameter range

0 - 0xFFFFFFFF

Default

N/A

Execution commands

The location where most AT commands set or query register values, execution commands execute an action on the device. Execution commands are executed immediately and do not require changes to be applied.

!R (Modem Reset)

Forces the cellular component to reboot.



CAUTION! This command is for advanced users, and you should only use it if the cellular component becomes completely stuck while in Bypass mode. Normal users should never need to run this command. See the [FR \(Force Reset\)](#) command instead.

Range

N/A

Default

N/A

IS (Force Sample)

When run, **IS** reports the values of all of the enabled digital and analog input lines. If no lines are enabled for digital or analog input, the command returns an error.

Command mode

In Command mode, the response value is a multi-line format, individual lines are delimited with carriage returns, and the entire response terminates with two carriage returns. Each line is a series of ASCII characters representing a single number in hexadecimal notation. The interpretation of the lines is:

- Number of samples. For legacy reasons this field always returns 1.
- Digital channel mask. A bit-mask of all I/O capable pins in the system. The bits set to **1** are configured for digital I/O and are included in the digital data value below. Pins D0 - D9 are bits 0 - 9, and P0 - P2 are bits 10 - 12.
- Analog channel mask. The bits set to **1** are configured for analog I/O and have individual readings following the digital data field.
- Digital data. The current digital value of all the pins set in the digital channel mask, only present if at least one bit is set in the digital channel mask.
- Analog data. Additional lines, one for each set pin in the analog channel mask. Each reading is a 10-bit ADC value for a 2.5 V voltage reference.

API operating mode

In API operating mode, **IS** immediately returns an **OK** response.

The API response is ordered identical to the Command mode response with the same fields present. Each field is a binary number of the size listed in the following table. Multi-byte fields are in big-endian byte order.

Field	Size
Number of samples	1 byte
Digital channel mask	2 bytes
Analog channel mask	1 byte
Samples	2 bytes each

Parameter range

N/A

Default

N/A

Operate in API mode

API mode overview	125
Use the AP command to set the operation mode	125
API frame format	125
Frame descriptions	129

API mode overview

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

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

Use the AP command to set the operation mode

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

AP command setting	Description
AP = 0	Transparent operating mode, UART serial line replacement with API modes disabled. This is the default option.
AP = 1	API operation.
AP = 2	API operation with escaped characters (only possible on UART).
AP = 3	N/A
AP = 4	MicroPython REPL
AP = 5	Bypass mode. This mode is for direct communication with the underlying chip and is only for advanced users.

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

API frame format

An API frame consists of the following:

- Start delimiter
- Length
- Frame data
- Checksum

API operation (AP parameter = 1)

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

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

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

API operation with escaped characters (AP parameter = 2)

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

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

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

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

Start delimiter field

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

Escaped characters in API frames

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

The following data bytes need to be escaped:

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

To escape a character:

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

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

Example: escape an API frame

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

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

You must escape the 0x13 byte:

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

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

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

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

Length field

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

Frame data

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

	Length		Frame data								
			Data								
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB		Data							

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

Multi-byte values are sent big-endian.

Calculate and verify checksums

To calculate the checksum of an API frame:

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

To verify the checksum of an API frame:

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

Example

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

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

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

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

Add these hex bytes:

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

Now take the result of 0x247 and keep only the lowest 8 bits which in this example is 0xC4 (the two far right digits). Subtract 0xC4 from 0xFF and you get 0x3B (0xFF - 0xC4 = 0x3B). 0x3B is the checksum for this data packet.

If an API data packet is composed with an incorrect checksum, the XBee Smart Modem will consider the packet invalid and will ignore the data.

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

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

Frame descriptions

The following sections describe the API frames.

AT Command - 0x08

Description

Use this frame to query or set parameters on the local device. Changes this frame makes to device parameters take effect after executing the AT command.

Format

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

Field name	Field value	Data type	Description
Frame type	0x08	Byte	
Frame ID		Byte	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
AT command		Byte	Command name: two ASCII characters that identify the AT command.
Parameter value		Byte	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the register.

AT Command: Queue Parameter Value - 0x09

Description

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

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

When querying parameter values, the 0x09 frame behaves identically to the 0x08 frame. The device returns register queries immediately and does not queue them. The response for this command is also an AT Command Response frame (0x88).

Format

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

Field name	Field value	Data type	Description
Frame type	0x09	Byte	
Frame ID		Byte	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
AT command		Byte	Command name: two ASCII characters that identify the AT command.
Parameter value		Byte	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the register.

Transmit (TX) Request: IPv4 - 0x20

Description

A TX Request message causes the device to transmit data in IPv4 format. A TX request frame for a new destination creates a network socket. After the network socket is established, data from the network that is received on the socket is sent out the device's serial port in the form of a Receive (RX) Packet frame.

Format

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

Field name	Field value	Data type	Description
Frame type	0x20	Byte	
Frame ID		Byte	Reference identifier used to match status responses. 0 disables the TX Status frame.
Destination address		32-bit big endian	
Destination port		16-bit big endian	
Source port		16-bit big endian	If the source port is 0 , the device attempts to send the frame data using an existing open socket with a destination that matches the destination address and destination port fields of this frame. If there is no matching socket, then the device attempts to open a new socket. If the source port is non-zero, the device attempts to send the frame data using an existing open socket with a source and destination that matches the source port, destination address, and destination port fields of this frame. If there is no matching socket, the TX Status frame returns an error.
Protocol		Byte	0 = UDP 1 = TCP
Transmit options		Byte bitfield	Bit fields are offset 0 Bit field 0 - 7. Bits 0, and 2-7 are reserved, bit 1 is not. BIT 1 = 1 - Terminate the TCP socket after transmission is complete 0 - Leave the socket open. Closed by timeout, see TM (IP Client Connection Timeout) . Ignore this bit for UDP packets. All other bits are reserved and should be 0 .
Payload		Variable	Data to be transferred to the destination, may be up to 1500 bytes.

AT Command Response - 0x88

Description

A device sends this frame in response to an AT Command (0x08) frame. Some commands send back multiple frames.

Format

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

Field name	Field value	Data type	Description
Frame type	0x88	Byte	
Frame ID		Byte	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
AT command		Byte	Command name: two ASCII characters that identify the AT command.
Status	##	Byte	0 = OK 1 = ERROR 2 = Invalid command 3 = Invalid parameter
Parameter value		Byte	Register data in binary format. If the register was set, then this field is not returned.

Transmit (TX) Status - 0x89

Description

Indicates the success or failure of a transmit operation.

Format

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

Field name	Field value	Data type	Description
Frame type	0x89	Byte	
Frame ID		Byte	Refers to the frame ID specified in a previous transmit frame
Status		Byte	Status code (see the table below)

The following table shows the status codes.

Code	Description
0x0	Successful transmit
0x21	Failure to transmit to cell network
0x22	Not registered to cell network
0x2c	Invalid frame values (check the phone number)
0x31	Internal error
0x32	Resource error (retry operation later)
0x74	Message too long
0x78	Invalid UDP port
0x79	Invalid TCP port
0x7A	Invalid host address
0x7B	Invalid data mode
0x80	Connection refused
0x81	Socket connection lost
0x82	No server
0x83	Socket closed
0x84	Unknown server
0x85	Unknown error

Modem Status - 0x8A

Description

Cellular component status messages are sent from the device in response to specific conditions.

Format

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

Field name	Field value	Data type	Description
Frame type	0x8A	Byte	
Status	##	Byte	0 = Hardware reset or power up 1 = Watchdog timer reset 2 = Registered with cellular network 3 = Unregistered with cellular network 0x0E = Remote Manager connected 0x0F = Remote Manager disconnected

Receive (RX) Packet: IPv4 - 0xB0

Description

The XBee Smart Modem uses this frame when it receives RF data on a network socket that is created by a TX request frame or configuring [C0 \(Source Port\)](#).

Format

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

Socket behavior

Supported sockets	137
Socket timeouts	137
Socket limits in API mode	137
Enable incoming TCP sockets in API mode	137
API mode behavior for outgoing TCP and SSL connections	137
API mode behavior for outgoing UDP data	138
API mode behavior for incoming TCP connections	138
API mode behavior for incoming UDP data	139
Transparent mode behavior for outgoing TCP and SSL connections	139
Transparent mode behavior for outgoing UDP data	139
Transparent mode behavior for incoming TCP connections	140
Transparent mode behavior for incoming UDP connections	140

Supported sockets

The XBee Smart Modem supports the following number of sockets:

- 7 maximum: 4 SSL sockets maximum.¹

Socket timeouts

The XBee Smart Modem implicitly opens the socket any time there is data to be sent, and closes it according to the timeout settings. The [TM \(IP Client Connection Timeout\)](#) command controls the timeout settings.

Socket limits in API mode

There are a fixed number of sockets that can be created which varies by device variant and protocol.

When a [Transmit \(TX\) Request: IPv4 - 0x20](#) frame is sent to the XBee Smart Modem for a new destination, it creates a new socket. The exception to this is when using the UDP protocol with the **C0** source port, which allows unlimited destinations on the socket created by **C0 (Source Port)**.

If no more sockets are available, the device sends back a [Transmit \(TX\) Status - 0x89](#) frame with a Resource Error.

The Resource Error resolves when an existing socket is closed.

An existing socket may be closed when the socket times out (see [TM \(IP Client Connection Timeout\)](#) and [TS \(IP Server Connection Timeout\)](#)) or when the socket is closed via a TX request with the CLOSE flag set.

Enable incoming TCP sockets in API mode

In API mode, you can enable incoming connections to the XBee Smart Modem.

1. To enable listening, set **C0 (Source Port)** to the value of the listening port.
2. To use TCP for client and server socket connections, set **IP (IP Protocol)** to **0x01**.

The listener allows multiple clients (incoming connections), up to the limit of the maximum number of sockets on the system.

When the XBee Smart Modem receives RF data on the port defined by **C0**, you get a [Receive \(RX\) Packet: IPv4 - 0xB0](#) with the incoming address and port.

If you want to communicate back to the incoming connection, use the [Transmit \(TX\) Request: IPv4 - 0x20](#) and enter the received address and port as the destination address and port, along with the listening (**C0**) local source port.

API mode behavior for outgoing TCP and SSL connections

Note SSL is not currently enabled for the LTE-M product.

To initiate an outgoing TCP or SSL connection to a remote host, send a [Transmit \(TX\) Request: IPv4 - 0x20](#) frame to the XBee Smart Modem's serial port specifying the destination address and destination port for the remote host; the data is optional and the source port is **0**.

¹ TCP socket is used for Remote Manager, so if you have Remote Manager enabled, subtract 1 socket from the values above.

If the connection is disconnected at any time, send a Transmit TX Request frame to trigger a new connection attempt.

To send data over this connection use the [Transmit \(TX\) Request: IPv4 - 0x20](#).

The device sends a [Transmit \(TX\) Status - 0x89](#) frame in reply to the Transmit TX Request indicating the status of the request. A status of **0** indicates the connection and/or data was successful and a non-zero value indicates a failure.

Any data received on the connection is sent out the XBee Smart Modem's serial port as a Receive RX frame.

A connection is closed when:

- The remote end closes the connection.
- No data is sent or received for longer than the socket timeout set by [TM \(IP Client Connection Timeout\)](#).
- A Transmit TX Request is sent with the CLOSE flag set.

API mode behavior for outgoing UDP data

To send a UDP datagram to a remote host, send a [Transmit \(TX\) Request: IPv4 - 0x20](#) frame to the XBee Smart Modem's serial port specifying the destination address and destination port of the remote host. If you use a source port of **0**, the device creates a new socket for the purpose of sending to the remote host. The XBee Smart Modem supports a finite number of sockets, so if you need to send to many destinations:

1. The socket must be closed after use.
- or
2. You must use the socket specified by the [C0 \(Source Port\)](#) setting.

To use the socket specified by the **C0** setting, in the Transmit TX request frame use a source port that matches the value configured for the **C0** setting.

The device sends a [Transmit \(TX\) Status - 0x89](#) frame in reply to the Transmit TX Request to indicate the status of the request. A status of **0** indicates the data was successfully sent out of the device and a non-zero value indicates a failure.

Any data received on the UDP socket is sent out the XBee Smart Modem's serial port as a [Receive \(RX\) Packet: IPv4 - 0xB0](#) frame.

A UDP socket is closed when:

- No data has been sent or received for longer than the socket timeout set by [TM \(IP Client Connection Timeout\)](#).
- A transmit TX Request is sent with the CLOSE flag set.

API mode behavior for incoming TCP connections

For incoming connections and data in API mode, the XBee Smart Modem uses the [C0 \(Source Port\)](#) and [IP \(IP Protocol\)](#) settings to specify the listening port and protocol used. The XBee Smart Modem does not currently support the SSL protocol for incoming connections.

When the **IP** setting is TCP the XBee Smart Modem allows multiple incoming TCP connections on the port specified by the **C0** setting. Any data received on the connection is sent out the XBee Smart Modem's serial port as a [Receive \(RX\) Packet: IPv4 - 0xB0](#) frame.

To send data from the device over the connection, use the [Transmit \(TX\) Request: IPv4 - 0x20](#) frame with the corresponding address fields received from the Receive RX frame. In other words:

- Take the source address, source port, and destination port fields from the Receive (RX) frame and use those respectively as:
- The destination address, destination port, and source port fields for the Transmit (TX) Request frame.

A connection is closed when:

- The remote end closes the connection.
- No data has been sent or received for longer than the socket timeout set by [TS \(IP Server Connection Timeout\)](#).
- A Transmit (TX) Request frame is sent with the CLOSE flag set.

API mode behavior for incoming UDP data

When the [IP \(IP Protocol\)](#) setting is UDP, any data sent from a remote host to the XBee Smart Modem's network port specified by the [C0 \(Source Port\)](#) setting is sent out the XBee Smart Modem's serial port as a [Receive \(RX\) Packet: IPv4 - 0xB0](#) frame.

To send data from the XBee Smart Modem to the remote destination, use the [Transmit \(TX\) Request: IPv4 - 0x20](#) frame with the corresponding address fields received from the Receive RX frame. In other words take the source address, source port, and destination port fields from the Receive (RX) frame and use those respectively as the destination address, destination port, and source port fields for the Transmit (TX) Request frame.

Transparent mode behavior for outgoing TCP and SSL connections

Note SSL is not currently enabled for the LTE-M product.

For Transparent mode, the [IP \(IP Protocol\)](#) setting specifies the protocol and the [DL \(Destination Address\)](#) and [DE \(Destination Port\)](#) settings specify the destination address used for outgoing data (UDP) and outgoing connections (TCP and SSL).

To initiate an outgoing TCP or SSL connection to a remote host, send data to the XBee Smart Modem's serial port. If [CI \(Protocol/Connection Indication\)](#) reports a value of **0**, then the connection was successfully established, otherwise the value of **CI** indicates why the connection attempt failed. Any data received over the connection is sent out the XBee Smart Modem's serial port.

A connection is closed when:

- The remote end closes the connection.
- No data has been sent or received for longer than the socket timeout set by [TM \(IP Client Connection Timeout\)](#).
- You make and apply a change to the **IP**, **DL**, or **DE**.

Transparent mode behavior for outgoing UDP data

To send outgoing UDP data to a remote host, send data to the XBee Smart Modem's serial port. If [CI \(Protocol/Connection Indication\)](#) reports a value of **0**, the data was successfully sent; otherwise, the value of **CI** indicates why the data failed to be sent.

The [RO \(Packetization Timeout\)](#) setting provides some control in how the serial data gets packetized before being sent to the remote host. The first send opens up a UDP socket used to send and receive data. Any data received by this socket is sent out the XBee Smart Modem's serial port.

Transparent mode behavior for incoming TCP connections

The [C0 \(Source Port\)](#) and [IP \(IP Protocol\)](#) settings specify the listening port and protocol used for incoming connections (TCP) and incoming data (UDP) in Transparent mode. SSL is not currently supported for incoming connections.

When the **IP** setting is TCP and there is no existing connection to or from the XBee Smart Modem, the device accepts one incoming connection. Any data received on the connection is sent out the XBee Smart Modem's serial port. Any data sent to the XBee Smart Modem's serial port is sent over the connection. If the connection is disconnected, it discards pending data.

Transparent mode behavior for incoming UDP connections

When the [IP \(IP Protocol\)](#) setting is UDP any data sent from a remote host to the XBee Smart Modem's network port specified by [C0 \(Source Port\)](#) is sent out the XBee Smart Modem's serial port. Any data sent to the XBee Smart Modem's serial port is sent to the network destination specified by the [DL \(Destination Address\)](#) and [DE \(Destination Port\)](#) settings. If the **DL** and **DE** settings are unspecified or invalid, the XBee Smart Modem discards data sent to the serial port.

Troubleshooting

This section contains troubleshooting steps for the XBee Smart Modem.

- Cannot find the serial port for the device 142
- Correct a macOS Java error144
- Unresponsive cellular component in Bypass mode 145
- Syntax error at line 1 145
- Network connection issues 145

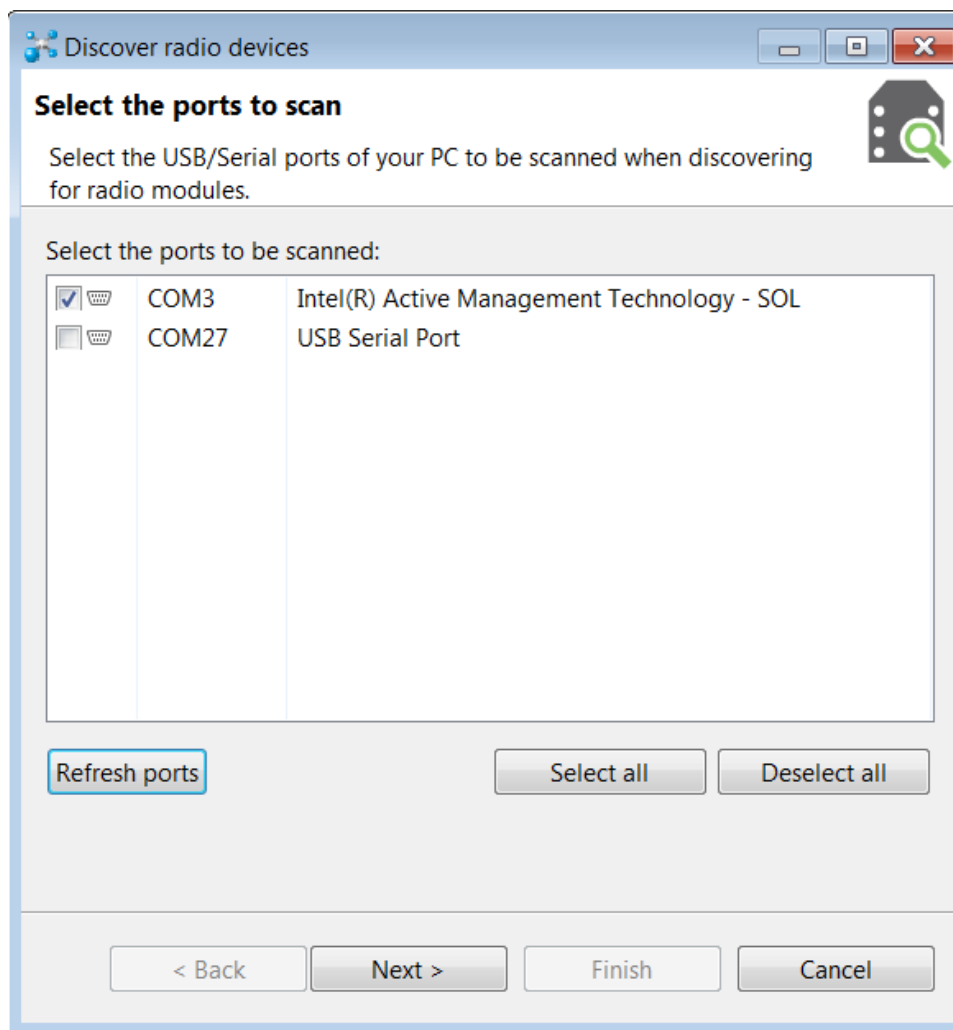
Cannot find the serial port for the device

Condition

In XCTU, the serial port that your device is connected to does not appear.

Solution

1. Click the **Discover radio modules** button .
2. Select all of the ports to be scanned.
3. Click **Next** and then **Finish**. A dialog notifies you of the devices discovered and their details.



4. Remove the development board from the USB port and view which port name no longer appears in the **Discover radio devices** list of ports. The port name that no longer appears is the correct port for the development board.

Other reasons that the XBee Smart Modem is not discoverable include:

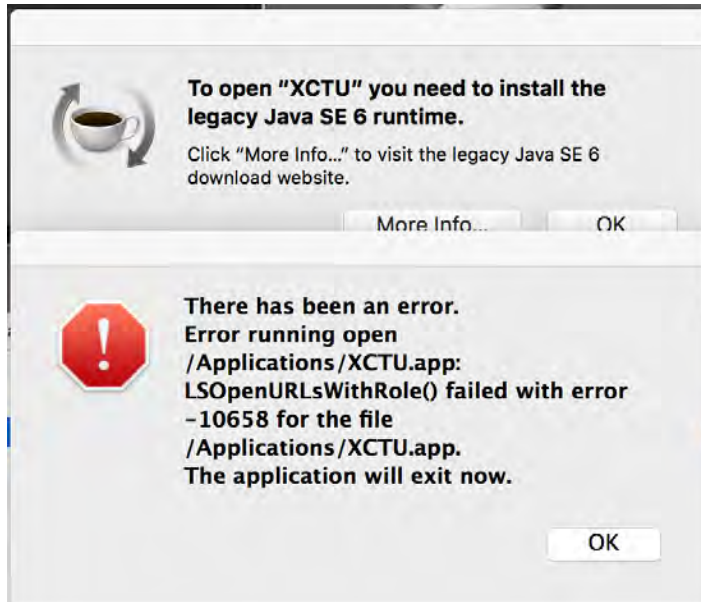
1. If you accidentally have the loopback pins jumpered.
2. You may not be using an updated FTDI driver.
 - a. This may require you to reboot your computer.
 - b. Disconnect the power and USB from the XBIB-U-DEV board and reconnect it.

Correct a macOS Java error

When you use XCTU on macOS computer, you may encounter a Java error.

Condition

When opening XCTU for the first time on a macOS computer, you may see the following error:



Solution

1. Click **More info** to open a browser window.
2. Click **Download** to get the file javaforosx.dmg.
3. Double-click on the downloaded javaforosx.dmg.
4. In the dialog, double-click the JavaForOSX.pkg and follow the instructions to install Java.

Unresponsive cellular component in Bypass mode

When in Bypass mode, the XBee Smart Modem does not automatically reset or reboot the cellular component if it becomes unresponsive.

Condition

In Bypass mode, the XBee Smart Modem does not respond to commands.

Solution

1. Query the [AI \(Association Indication\)](#) parameter to determine whether the cellular component is connected to the XBee Smart Modem software. If **AI** is **0x2F**, Bypass mode should work. If not, look at the status codes in [AI \(Association Indication\)](#) for guidance.
2. You can send the [!R \(Modem Reset\)](#) command to reset only the cellular component.

Syntax error at line 1

You may get a **syntax error at line 1** error after pasting example MicroPython code and pressing **Ctrl+D**.

Solution

This commonly happens when you accidentally type a character at the beginning of line 1 before pasting the code.

Network connection issues

Condition

The XBee Smart Modem is not joining the network, [AI \(Association Indication\)](#) is cycling between **0xFF** (Initializing), **0x22** (Registering to Cellular Network) and **0x25** (Cellular Network Registration Denied).

Solution

Some things to check are:

- The antennas are connected correctly to the device.
- The SIM card is seated properly in the device.
- APN is set correctly (see below).

Set the APN value

Note The following instructions are not indicative of the final product. In the full release the APN commands will be in place to set this value without having to use Bypass operating mode.

To check and set the currently active APN string you must perform the following steps:

1. [Enter Bypass operating mode.](#)
2. Type **at+cgdcont=1,"IP","<APN String>"** and press **Enter**. For example for this early adopter kit:
at+cgdcont=1,"IP","vzwinternet"
3. [Leave Bypass operating mode.](#)

Regulatory information

United States (FCC)	148
IC (Industry Canada)	150

United States (FCC)

XBee Smart Modems comply with Part 15 of the FCC rules and regulations. Compliance with the labeling requirements, FCC notices and antenna usage guidelines is required.

To fulfill FCC Certification, the OEM must comply with the following regulations:

1. The system integrator must ensure that the text on the external label provided with this device is placed on the outside of the final product.
2. RF Modules may only be used with antennas that have been tested and approved for use with the modules.

OEM labeling requirements



WARNING! As an Original Equipment Manufacturer (OEM) you must ensure that FCC labeling requirements are met. You must include a clearly visible label on the outside of the final product enclosure that displays the following content:

Required FCC Label for OEM products containing the XBee3 Cellular LTE-M RF Module

Contains FCC ID: MCQ-XB3M1

Contains FCC ID: XPY2AGQN4NNN

The enclosed device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (i.) this device may not cause harmful interference and (ii.) this device must accept any interference received, including interference that may cause undesired operation.

FCC notices

IMPORTANT: XBee3 RF Modules have been certified by the FCC for use with other products without any further certification (as per FCC section 2.1091). Modifications not expressly approved by Digi could void the user's authority to operate the equipment.

IMPORTANT: OEMs must test final product to comply with unintentional radiators (FCC section 15.107 & 15.109) before declaring compliance of their final product to Part 15 of the FCC Rules.

IMPORTANT: The RF module has been certified for remote and base radio applications. If the module will be used for portable applications, the device must undergo SAR testing.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: Re-orient or relocate the receiving antenna, Increase the separation between the equipment and receiver, Connect equipment and receiver to outlets on different circuits, or Consult the dealer or an experienced radio/TV technician for help.

FCC-approved antennas

The XBee3 RF Module can be installed using antennas and cables constructed with non-standard connectors (RPSMA, RPTNC, and so forth) An adapter cable may be necessary to attach the XBee connector to the antenna connector.

The modules are FCC approved for fixed base station and mobile applications for the channels indicated in the tables below. If the antenna is mounted at least 20 cm (7.87 in) from nearby persons, the application is considered a mobile application.

The antennas below have been approved for use with this module. Digi does not carry all of these antenna variants. Contact Digi Sales for available antennas.

Bluetooth antennas

The following tables cover the antennas that are approved for use with the Bluetooth radio.

Integral antenna

Part number	Type (description)	Gain	Application
31000020-02	Integral antenna	-2.5 dBi	Fixed/Mobile

Dipole antennas

Part number	Type (description)	Gain	Application
29000095	Dipole (Half-wave articulated RPSMA - 4.5")	2.1 dBi	Fixed/Mobile
A24-HASM-450	Dipole (Half-wave articulated RPSMA-4.5")	2.1 dBi	Fixed/Mobile
A24-HABSM	Dipole (Articulated RPSMA)	2.1 dBi	Fixed
A24-HABUF-P5I	Dipole (Half-wave bulkhead mount U.FL w/ 5" pigtail)	2.1 dBi	Fixed
A24-HASM-525	Dipole (Half-wave articulated RPSMA-5.25")	2.1 dBi	Fixed/Mobile

Flex PCB antennas

Part number	Type (description)	Gain	Application
29000812	Flexible PCB, U.FL w/ 200mm pigtail	4.4 dBi	Fixed/Mobile
FXP74.07.0100A	Flexible PCB, U.FL w/ 100mm pigtail	4.0 dBi	Fixed/Mobile

Cellular antennas

The gain of the system antenna (i.e. the combined transmission line, connector, cable losses and radiating element gain) must not exceed the values below for mobile and fixed or mobile operating configurations:

- 3.67 dBi in 700 MHz, i.e. LTE FDD-12 band
- 4.10 dBi in 850 MHz, i.e. LTE FDD-5 band

- 6.74 dBi in 1700 MHz, i.e. LTE FDD-4 band
- 7.12 dBi in 1900 MHz, i.e. LTE FDD-2 band

RF exposure

If you are integrating the XBee3 RF Module into another product, you must include the following Caution statement in OEM product manuals to alert users of FCC RF exposure compliance:



CAUTION! To satisfy FCC RF exposure requirements for mobile transmitting devices, a separation distance of 20 cm or more should be maintained between the antenna of this device and persons during device operation. To ensure compliance, operations at closer than this distance are not recommended. The antenna used for this transmitter must not be co-located in conjunction with any other antenna or transmitter.

IC (Industry Canada)

Labeling requirements

Labeling requirements for Industry Canada are similar to those of the FCC. A clearly visible label on the outside of the final product enclosure must display the following text.

Contains IC: 1846A-XB3M1

Contains IC: 8595A-2AGQN4NNN

The integrator is responsible for its product to comply with IC ICES-003 & FCC Part 15, Sub. B - Unintentional Radiators. ICES-003 is the same as FCC Part 15 Sub. B and Industry Canada accepts FCC test report or CISPR 22 test report for compliance with ICES-003.

This device complies with Industry Canada licence-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes: (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

RF Exposure



CAUTION! This equipment is approved for mobile and base station transmitting devices only. Antenna(s) used for this transmitter must be installed to provide a separation distance of at least 20 cm from all persons and must not be co-located or operating in conjunction with any other antenna or transmitter.



ATTENTION! Cet équipement est approuvé pour la mobile et la station base dispositifs d'émission seulement. Antenne(s) utilisé pour cet émetteur doit être installé pour fournir une distance de séparation d'au moins 20 cm à partir de toutes les personnes et ne doit pas être situé ou fonctionner en conjonction avec tout autre antenne ou émetteur.

Transmitters with Detachable Antennas

This radio transmitter has been approved by Industry Canada to operate with the antenna types listed in [FCC-approved antennas](#) with the maximum permissible gain and required antenna impedance for each antenna type indicated. Antenna types not included in this list, having a gain greater than the maximum gain indicated for that type, are strictly prohibited for use with this device.

Le présent émetteur radio a été approuvé par Industrie Canada pour fonctionner avec les types d'antenne énumérés ci-dessous et ayant un gain admissible maximal et l'impédance requise pour chaque type d'antenne. Les types d'antenne non inclus dans cette liste, ou dont le gain est supérieur au gain maximal indiqué, sont strictement interdits pour l'exploitation de l'émetteur.

Detachable Antenna

Under Industry Canada regulations, this radio transmitter may operate using only an antenna of a type and maximum (or lesser) gain approved for the transmitter by Industry Canada. To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (EIRP) is not more than that necessary for successful communication.

Conformément à la réglementation d'Industrie Canada, le présent émetteur radio peut fonctionner avec une antenne d'un type et d'un gain maximal (ou inférieur) approuvé pour l'émetteur par Industrie Canada. Dans le but de réduire les risques de brouillage radioélectrique à l'intention des autres utilisateurs, il faut choisir le type d'antenne et son gain de sorte que la puissance isotrope rayonnée équivalente (p.i.r.e.) ne dépasse pas l'intensité nécessaire à l'établissement d'une communication satisfaisante.