APPLICATION NOTE

AN413

ZigBee™/802.15.4 Application Kit

Introduction

The ZigBee™/802.15.4 Application Kit combines MaxStream's XBee™ RF modem with a popular Rabbit-Core module. The XBee™ modem is mounted on an RF Interface module that provides the power supply and an RS-232 serial interface with flow control. Three RF Interface modules are included in the Application Kit along with three XBee™ RF modems to allow you to use one device, the "coordinator," attached to the Rabbit-based control system, and to place the other two devices, the "children," up to 100 ft (30 m) away.

The ZigBee™/802.15.4 Application Kit serves as a template for a Rabbit-based system that integrates XBee™ wireless modems with your Rabbit-based systems. The sample programs included with the Application Kit serve as a template for wireless applications where low power and low data rates are needed. Applications include simple remote monitoring, proximity sensor readings, wireless I/O control, and data transmission.

A user-configurable interface allows you to set up the network, identify and poll the network for other similar XBee™ RF modems, and allows you to control LEDs and switches via an RF Interface module with an XBee™ RF modem that serves as the "coordinator."

Features

- RCM3720 RabbitCore module with RCM3720 Prototyping Board
- RF Interface modules with MaxStream XBee™ RF modems
- Complete Dynamic C software CD and supplemental CD with sample programs and reference information related to the Application Kit

Example Applications

- Low-cost wireless embedded control applications
- Remote monitoring of equipment, devices, locations
- Simple data-logging applications
- Mesh networking control

What Else You Will Need

Besides what is supplied with the Application Kit, you will need a PC with an available COM or USB port to program the RCM3720 and the XBee™ RF modems in the Application Kit. If your PC only has a USB port, you will also need an RS-232/USB converter. Note that not all RS-232/USB converters work with Dynamic C and the XBee™ RF modem firmware. Your PC also needs an RJ-45 jack to allow an Ethernet interface with the RCM3720, and you will need a CAT 5/6 Ethernet cable for an Ethernet connection between your PC and the RCM3720.

Hardware Setup

The ***ZigBee™/802.15.4 Application Kit Getting Started*** instructions included with the Application Kit show how to set up and program the RCM3720, the RF Interface module, and the XBee™ RF modem. Figure 1 shows how the RF Interface module and the XBee™ RF modem interface with the RCM3720 and the RCM3720 Prototyping Board.

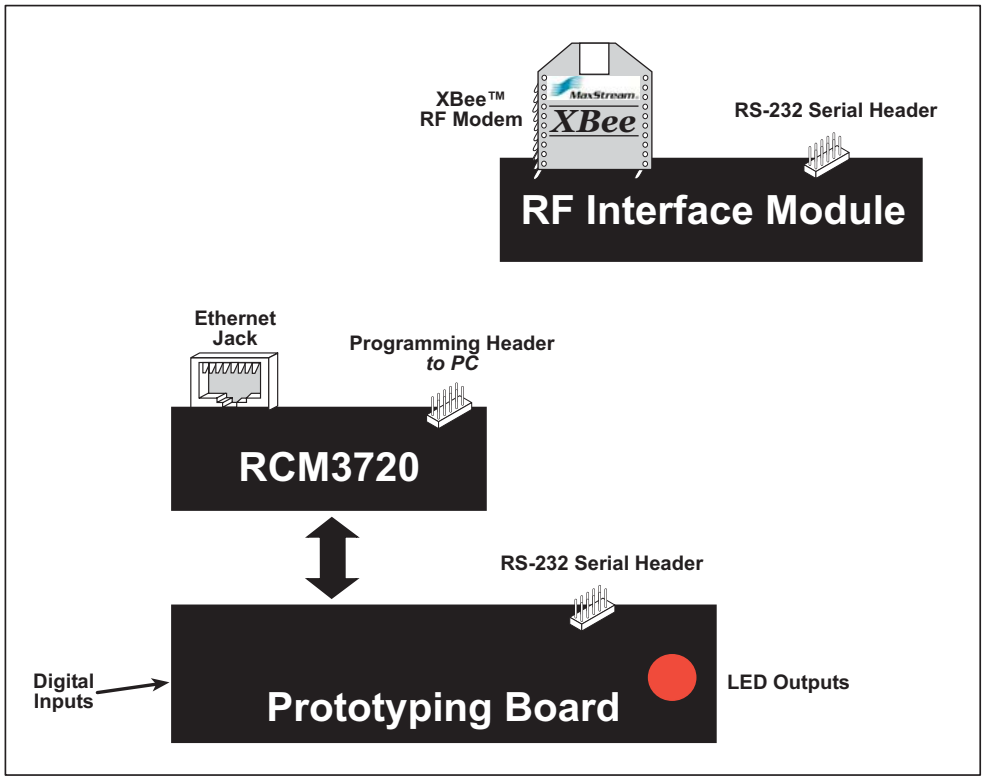


Figure 1. ZigBee™/802.15.4 Kit Setup

One RF Interface module is connected to the RCM3720 Prototyping Board via a ribbon cable, and the remaining two RF interface modules with XBee™ RF modems installed may be placed up to 100 ft (30 m) away.

The RF Interface modules with XBee™ RF modems may be set up in either a *peer-to-peer* network or a *mesh* network. Firmware is supplied for both options.

- Peer-to-peer network — XBee™ 802.15.4 firmware.
- Mesh network —
    - XBee™ ZigBee™ coordinator API commands (not supported by Dynamic C)
    - XBee™ ZigBee™ coordinator AT commands
    - XBee™ ZigBee™ router API commands (not supported by Dynamic C)
    - XBee™ ZigBee™ router AT commands

While as many as six “children” may be nested per “parent” router in a mesh network up to six layers deep, Dynamic C can only support up to 64 devices.

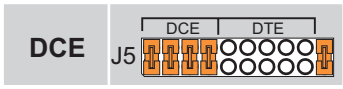
The following steps from the ***ZigBee™/802.15.4 Application Kit Getting Started*** instructions summarize the setup process once Dynamic C and the software from the supplemental CD have been installed on your PC.

RF Interface Module Connections

Set up all three RF Interface modules as explained in the ***ZigBee™/802.15.4 Application Kit Getting Started*** instructions.

1. Snap in four standoffs to the four holes at the corners on the bottom side of the RF Interface module.
2. Install the XBee™ RF modem into the inside-facing sockets of header sockets J1 and J2 on the RF Interface module. Press the XBee™ modem's pins firmly into the RF Interface module header sockets.

   The outside-facing sockets of header sockets J1 and J2 on the RF Interface module carry the identical signals that are being used by the XBee™ RF modem pin to the side of the sockets. This allows convenient access to the XBee™ RF modem pins for development and diagnostic purposes.
3. Install 5 jumpers on header J5 for the DCE configuration as shown.



4. Place 3 AAA batteries in a battery holder, then connect the red wire from the battery holder to the + terminal of screw terminal header J3, and connect the black wire from the battery holder to the – terminal of screw terminal header J3.

   If the battery holder has an on/off switch on the opposite side, make sure the switch is in the **ON** position. If there is no on/off switch, you may remove a battery to turn the RF Interface module off.

   You may use your own power supply with an output of 3.5–6.0 V DC.
5. Use the programming cable to connect the RF Interface module to your PC to download the firmware. Connect the 10-pin connector of the programming cable (the RCM3720 adapter board must be removed) to header J6 on the RF Interface module. Line up the colored edge of the programming cable with pin 1 of header J6 as shown. The DB9 connector end of the programming cable is attached to a COM (serial) port on the PC.

It is recommended that you assign a unique "name" up to 20 characters long to each RF Interface module, and place a sticker on the module to identify it. These "names" will be "assigned" to the XBee™ RF modems when you configure the networking parameters.

You will have one "coordinator" device and two "children." The "coordinator" RF Interface module will be connected to the RS-232 header on the Prototyping Board, and the "children" can be up to 100 ft (30 m) away.

> **NOTE:** The "coordinator" is named DIO-COORD in the sample programs.

Install the XBee™ RF Modem Firmware

1. Locate and double-click **`Setup_x-ctu_5013.exe`** in the Dynamic C **`DCRabbit…\xbee firmware`** directory to install the X-CTU application that you will use to download the firmware.
2. Start X-CTU from the desktop icon and set the "PC Settings" tab to **9600** baud, **HARDWARE** flow control, **8** data bits, parity **NONE**, **1** stop bit.
3. Under the "Modem Configuration" tab click the "Download new versions…" button, select "File," and browse the Dynamic C **`DCRabbit…\xbee firmware`** directory to select the ZIP file corresponding to your network. There are two firmware options, depending on whether you will be using a peer-to-peer (802.15.4, NonBeacon) network or a mesh (ZigBee™) network (**`XBee_1yyy.zip`** for a peer-to-peer network, or **`XBee_8yyy.zip`** for a mesh network, where **`yyy`** specifies the version number).
4. Select the ZIP file, click "Open," "OK," then "Done."

You are now ready to load the three XBee™ RF modems with the new firmware. Repeat the remaining steps for each of the three RF Interface modules with their XBee™ RF modems installed and the programming cable connected to header J6 of the RF Interface modules. Remember to set the jumpers on header J5 for the DCE configuration shown above.

1. Under the "Modem Configuration" tab choose "XB24" from the "Modem" pull-down menu.
2. Choose "XBEE 802.15.4" or the "COORDINATOR AT"/"ROUTER AT" from the "Function Set" pull-down menu.
3. Choose "1yyy" or "8yyy" from the "Version" pull-down menu ("1yyy" or "8yyy" matches the version number in the file name). The "8yyy" versions have options depending on whether you are configuring the coordinator or a router.
4. Check the "Always update firmware" box.
5. Click "Write" to load the XBee™ RF modem with the new firmware.

There are networking and I/O settings that also need to be set via the "Modem Configuration" tab of the X-CTU application. Before changing any configurations, it is a good idea to select the desired firmware version, then press the "Show Defaults" button to set all the values to their default states. Here are the most likely settings you will then configure.

Networking & Security

ID - Pan ID = 0x0000 – 0xFFFF (peer-to-peer network);
0x0000 – 0x3FFF, 0xFFFF to assign randomly (mesh network)

DH - Destination Address High and DL - destination Address Low must be set to 0 (peer-to-peer network)

CE - Coordinator Enable = 0 end device; 1 coordinator (peer-to-peer network only)

A2 - Coordinator Association = choose a value 0–7 from dropdown list (peer-to-peer network only)

NI - Node ID = up to a 20-character ASCII string

I/O Settings

D4–D0 - DIO4–DIO0 Configuration = configure I/O from dropdown list (peer-to-peer network children only)

IC - DIO Change Detect = 0 disabled; 1 enabled (peer-to-peer network only)

I/O Line Passing

IA - I/O Input Address = 0x0000 – 0xFFFF (peer-to-peer network)

IR - Sample Rate = 0x0000 – 0xFFFF (peer-to-peer network)

Click "Write" to load the XBee™ RF modem with the configurations.

Additional information on configurations is provided in the ***XBee™ 802.15.4 Protocol Manual*** (peer-to-peer networks) and the ***XBee™ ZigBee™ Protocol Manual*** (mesh networks).

RCM3720 Module Connections

Turn the RCM3720 module so that the Ethernet jack is on the left. Insert the module's J1 header into the J5 socket on the Prototyping Board. The shaded corner notch at the bottom right corner of the RCM3720 module should face the same direction as the corresponding notch below it on the Prototyping Board.

> **NOTE:** It is important that you line up the pins on header J1 of the RCM3720 module exactly with the corresponding pins of the J5 socket on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the module will not work. Permanent electrical damage to the module may also result if a misaligned module is powered up.

Press the module's pins firmly into the Prototyping Board socket.

The programming cable with the RCM3720 adapter board connects the RCM3720 to the PC running Dynamic C to download programs and to monitor the RCM3720 module during debugging.

Attach the DB9 connector end of the programming cable to a COM (serial) port on the PC. Dynamic C uses a COM port to communicate with the target system. The default selection is COM1, but you may select a different COM port when you install or run Dynamic C.

Connect the 10-pin connector of the programming cable and adapter board to header J2 on the RCM3720 module. The adapter board converts the PC voltage to the voltage on the RCM3720. Orient the programming cable and adapter board so that the colored edge of the programming cable lines up with the dot on the adapter board and pin 1 on the RCM3720.

Connect the other end of the programming cable to a COM port on your PC.

> **NOTE:** Some PCs now come equipped only with a USB port. It may be possible to use an RS-232/USB converter (Part No. 540-0070) with the programming cable supplied with this Application Kit. Note that not all RS-232/USB converters work with Dynamic C.

When all other connections have been made, you can connect power to the Prototyping Board. Connect the wall transformer to 3-pin header J1 on the Prototyping Board. The connector may be attached either way as long as it is not offset to one side—the center pin of J1 is always connected to the positive terminal, and either edge pin is negative.

Plug in the wall transformer. The power LED beside the **RESET** button on the Prototyping Board should light up. The RCM3720 and the Prototyping Board are now ready to be used.

> **NOTE:** The **RESET** button is provided on the Prototyping Board to allow a hardware reset without disconnecting power.

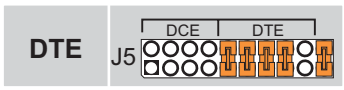
Alternate Power-Supply Connections

The 3-pin connector allows you to connect your own power supply—connect the center pin to the positive terminal, and connect either edge pin to the negative terminal. The power supply should deliver at least 200 mA at 7.5 V–15 V DC.

Set Up RF Interface Modules and Prototyping Board

Use a 10-pin to 10-pin IDC serial cable with a 0.1" pitch to connect header J6 of the "coordinator" RF Interface module to header J3, the RS-232 header on the RCM3720 Prototyping Board. Line up the red colored edge with pin 1 on both boards. Place the "child" RF Interface modules up to 100 ft (30 m) away.

Set up the jumpers on header J5 of the RF Interface module for DTE operation now that the firmware is loaded.



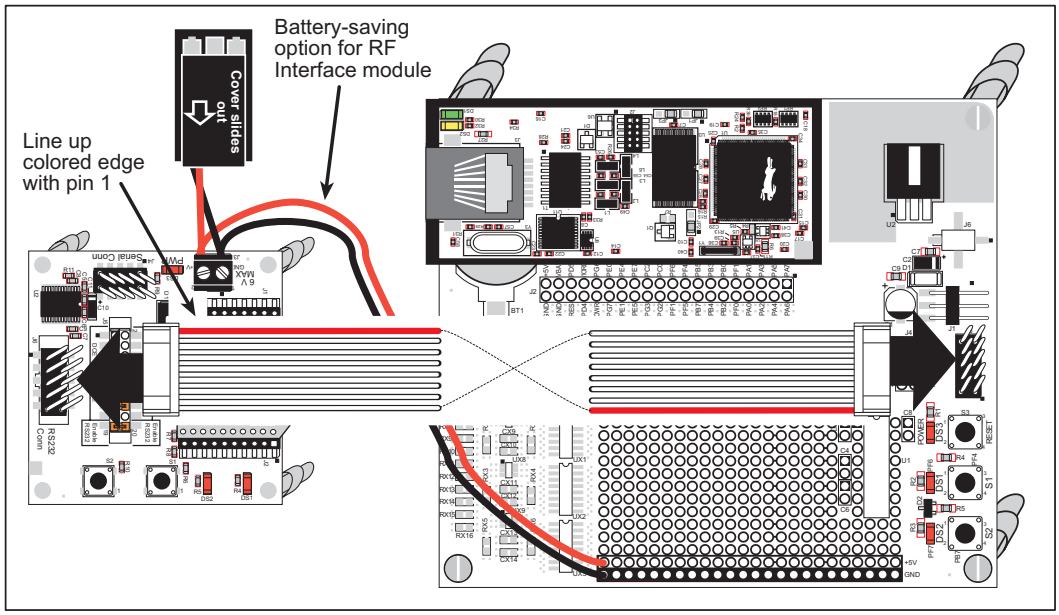


Figure 2. Serial Cable Connection RF Interface Module to Prototyping Board

Alternative Serial Cable Connections

The serial signals with handshaking are available on two headers on the RF Interface module.

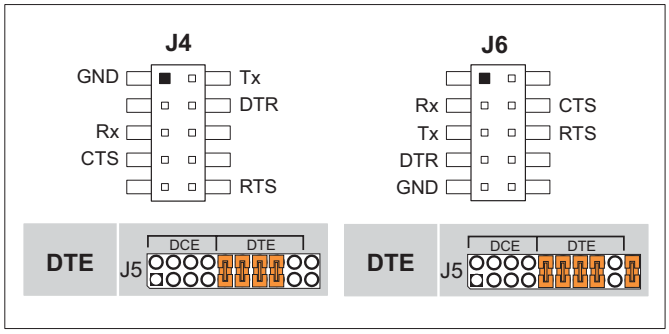


Figure 3. Serial Header J4 and J6 Pinouts and Configuration

- Header J6 provides regular RS-232 level signals on a 2 × 5 IDC header with a 0.1" pitch. You may interface this header to any Rabbit-based board and set up the serial and flow control signals with macros as shown in the sample programs.
- Header J4 provides TTL or CMOS level signals on a 2 × 5 IDC header with a 1.27 mm pitch. These signals will interface with an SF1000 location on a suitably equipped Rabbit Semiconductor board such as the BL2600 or the RCM3300 Prototyping Board. Disable the RS-232 transceiver when you use this option by removing the jumper across pins 19–20 on header J5 of the RF Interface module.

Figure 4 shows a sample connection using the 1.27 mm serial cable and connector adapter (from in the bag of parts) between header J4 on the RF Interface module and the SF1000 connector (header J11) on the RCM3300 Prototyping Board.

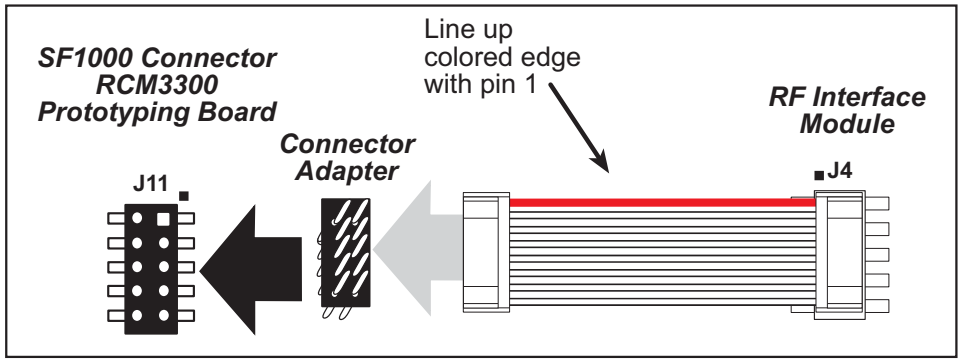


Figure 4. 1.27 mm Serial Cable Connection Showing Use of Connector Adapter with SF1000 Header on RCM3300 Prototyping Board

The **Serial Port Macros** section describes how to set the macros in the sample programs to use the SF1000 connector on the RCM3300 Prototyping Board.

Other Options

Firmware Download Options

Only the firmware supplied on the supplemental CD with this Application Kit has been tested with Dynamic C and the sample programs. Later versions of the firmware, which are available from Max-Stream, have not been tested.

Header J5 on the RF Interface module may be configured to download firmware to the XBee™ RF modem directly from a Rabbit-based board, but this option is not supported at this time.

RF Modem Options

The XBee™ RF modems are pin-for-pin interchangeable with the XBee-PRO™ RF modems. Both modems are available from MaxStream.

RF Interface Modules

Individual RF Interface modules with the battery holder and batteries may also be purchased from Rabbit Semiconductor. Remember that you will also need an RF modem — XBee™ and XBee-PRO™ RF modems are available from MaxStream.

Sample Programs

The following sample programs are available for the ZigBee™/802.15.4 Application Kit, and can be found in the Dynamic C **SAMPLES\ZIGBEE** folder.

In order to run these and other sample programs,

1. Your RCM3720 must be plugged in to the Prototyping Board as described in the ***ZigBee™/802.15.4 Application Kit Getting Started*** instructions.
2. Dynamic C and the software from the supplemental CDs in the ZigBee™/802.15.4 Application Kit must be installed and running on your PC.
3. The programming cable must connect the programming header on the RCM3720 to your PC.
4. Power must be applied to the RCM3720 via the RCM3720 Prototyping Board.
5. The XBee™ RF modems must be installed on the RF Interface modules, which must be powered up, and the appropriate firmware and any configurations must have been loaded to the XBee™ RF modems via the X-CTU application while the jumpers on header J5 of the RF Interface module were in the DCE position.
6. The jumpers on header J5 of the RF Interface module must be in the DTE position to run the sample program, and the coordinator RF Interface module must be connected via a serial ribbon cable to the RS-232 header on the Prototyping Board.

To run a sample program, open it with the **File** menu, then run it by selecting **Run** in the **Run** menu (or press **F9**).

Before changing any firmware configurations for the XBee™ RF modems, start the X-CTU application, then select the desired firmware version under the "Modem Configuration" tab, then press the "Show Defaults" button to set all the values to their default states.

Peer-to-Peer Network Sample Programs

- **`XB_BASIC_DIO.C`**—This sample program constantly discovers nodes and turns LEDs DS1 and DS2 on the RF Interface module on and off. The pushbuttons S1 and S2 and the battery voltage are also monitored and their status is displayed in the Dyamic C **STDIO** window.

  The digital I/Os for the XBee™ RF modems are set up to match the RF Interface module:

  DIO0 = Output DS1 LED
  DIO1 = Output DS2 LED
  DIO2 = Input S1 pushbutton switch
  DIO3 = ADC BAT battery voltage monitor
  DIO4 = Input S2 pushbutton switch

  Before running this sample program, make sure that you have loaded the peer-to-peer network firmware on all three XBee™ RF modems. The XBee™ RF modem on the RF Interface module connected to the Prototyping Board is the coordinator.After you have loaded the peer-to-peer firmware on the coordinator, set up the following parameters with the X-CTU application (the jumpers on J5 are in the DCE position at this time — set them to DTE after all the configuring has been completed).

  Networking & Security

  ID - Pan ID = 0xAAAA
  CE - Coordinator Enable = 1
  A2 - Coordinator Association = 6
  NI - Node ID = DIO-COORD

  I/O Settings

  IC - DIO Change Detect = 1 (enabled)

  I/O Line Passing

  IA - I/O Input Address = 0xFFFF

  Everything else remains at default settings

  Click “Write” to load the XBee™ RF modem with the configurations.

  The remaining “children” XBee™ RF modems on the RF Interface modules are nodes, and must have the following parameters set up with the X-CTU application (the jumpers on J5 are in the DCE position at this time — set them to DTE after all the configuring has been completed):

  Networking & Security

  ID - Pan ID = 0xAAAA
  A1 - End Device Association = 6
  NI - Node ID = any 20-byte ASCII name

  I/O Settings

  D4 - DIO4 Configuration = 3 (input)
  D3 - DIO3 Configuration = 2 (ADC)
  D2 - DIO2 Configuration = 3 (input)
  D1 - DIO1 Configuration = 4 (output low)
  D0 - DIO0 Configuration = 4 (output low)
  IC - DIO Change Detect = 1 (enabled)
  IR - Sample Rate = 0xFFFF (~every 65 seconds)

  I/O Line Passing

  IA - I/O Input Address = 0xFFFF
  T0 - D0 Output Timeout = 0
  T1 - D1 Output Timeout = 0

  Everything else remains at default settings

  Click “Write” to load the XBee™ RF modem with the configurations.

  The sample program may now be run.

- **XB_RWEB_DIO.C**—The Dynamic C RabbitWeb module (included with this Application Kit) is required to run this sample program. Configure the networking and I/O settings for the XBee™ RF modems exactly as described for the **XB_BASIC_DIO.C** sample program. The sample program may now be run.

  The **XB_RWEB_DIO.C** sample program allows you to control, configure, and monitor a peer-to-peer network via the Ethernet using a Web browser. There are several pages:

  a *settings page* that allows you to change the serial COM port setting for the Rabbit processor

  a *configuration page* that allows you to configure the coordinator XBee™ RF modem and to set up the peer-to-peer network

  a *node discovery* page that allows you to discover nodes that have associated with the peer-to-peer network — this page includes detailed information such as the 64-bit address, signal strength, I/O configuration, and battery status for nodes that were detected — and allows you to control the digital I/O

  a *terminal page* that allows you to send AT commands directly to the coordinator XBee™ RF modem and send and receive raw ASCII data to any remotely associated node.

  Use auto-discovery to find all associated nodes. Once all nodes are found, uncheck the auto-discovery check box and check the auto DIO Refresh check box. Then click DIO REFRESH to monitor all the associated nodes' digital I/Os and battery voltages.

Mesh Network Sample Programs

Before you run the mesh network sample programs, the Dynamic C option to enable separate instruction and data spaces must be checked or enabled. This option is set under **Project Options** in the "Compiler" tab.

- **XB_BASIC_MESH.C**—This sample program will constantly monitor the personal area network (PAN) for associated nodes, and will display the node information to the Dynamic C **STDIO** window.

  Before running this sample program, you will have to load the appropriate mesh network firmware on all three XBee™ RF modems. The XBee™ RF modem on the RF Interface module connected to the Prototyping Board is the coordinator.After you have loaded the XBEE COORDINATOR AT firmware on the coordinator, set up the following parameters with the X-CTU application (the jumpers on J5 are in the DCE position at this time — set them to DTE after all the configuring has been completed).

  Networking & Security

  ID - Pan ID = 0x3333

  Everything else remains at default settings

  Click "Write" to load the XBee™ RF modem with the configurations.

  The remaining "children" XBee™ RF modems on the RF Interface modules are nodes, and must have the following parameters set up with the X-CTU application (the jumpers on J5 are in the DCE position at this time — set them to DTE after all the configuring has been completed):

  Networking & Security

  ID - Pan ID = 0x3333

  NI - Node ID = any 20-byte ASCII name

  Everything else remains at default settings

  Click "Write" to load the XBee™ RF modem with the configurations.

  NOTE: To change the Node ID on an XBee™ RF modem, you must use the X-CTU "Terminal" window and use the AT command NI. For example, to name a node "Rabbit_1," enter the string **AT NI Rabbit_1**. Then execute the **AT WR** command to save the node ID to nonvolatile memory.

  The sample program may now be run.

  To help demonstrate a mesh network, start running the sample program with only the coordinator powered up. Then power on one router node. Wait for the node to associate, and then power on the second router node. When the second node associates, it will be a "child" node to the first associated router because right after the first associated nodes are found the coordinator sets the Node Join time (NJ) to 0, which does not allow any more nodes to associate directly to the coordinator.This causes all new nodes to associate through routers. Now if the "parent" router is powered down, the coordinator will reset the network using the NR command with a parameter 1, which sends a broadcast transmission to reset the network layer parameters on all nodes in the personal area network. This will allow orphaned nodes to find a new parent and re-associate.

- **XB_RWEB_MESH.C**—The Dynamic C RabbitWeb module (included with this Application Kit) is required to run this sample program. Configure the networking and I/O settings for the XBee™ RF modems exactly as described for the **XB_BASIC_MESH.C** sample program. The sample program may now be run.

  The **XB_RWEB_MESH.C** sample program allows you to control, configure, and monitor a mesh network via the Ethernet using a Web browser. There are several pages:

  a *settings page* that allows you to change the serial COM port setting for the Rabbit processor

  a *configuration page* that allows you to configure the coordinator XBee™ RF modem and to set up the peer-to-peer network

  a *terminal page* that allows you to send AT commands directly to the coordinator XBee™ RF modem and send and receive raw ASCII data to any remotely associated node.

  a *node discovery* page that allows you to discover nodes that have associated with the mesh network — the following information is reported for each discovered node:

  MY (my id)

  SH (serial high)

  SL (serial low)

  NI (node id)

  PN (parent network)

  DT (device type) (1 Byte: 0 = coordinator, 1 = router, 2 = end device)

  ST (status)

  PR (profile id)

  MI (manufacturer id)

Appendix — Software Reference

Sample Program

Let's examine some of the code in the **XB_BASIC_DIO.C** sample program.

First, the program settings used at startup are defined. These macros are used by the parameters in the function calls, and you may change the macro definitions to suit your needs.

Serial Port Macros

```
#define ATCMDRSP_SP D   //set to serial port A, B, C, D, E, or F
#define DINBUFSIZE 511     // PC1 = RxD -- Xbee pin 2 = Dout
#define DOUTBUFSIZE 127    // PC0 = TxD -- Xbee pin 3 = Din
#define SERD_RTS_PORT PCDR // RTS is output flow control
#define SERD_RTS_SHADOW PCDRShadow
#define SERD_RTS_BIT 2     // PC2
#define SERD_CTS_PORT PCDR // CTS is input flow control
#define SERD_CTS_BIT 3     // PC3
#define DEFAULTBAUD 9600L  // xbee factory default baud rate
```

The **ATCMDRSP_SP** macro is used to specific the serial port used by AT commands in the Dynamic C **ATCMDRSP.LIB** library. The **D** specifies Serial Port D as the main RS-232 serial port.

The **DINBUFSIZE** macro is specifies the size of the buffer used to store data received from pin 2 of the XBee™ RF modem, which was configured for DOUT. The **D** in **DINBUFSIZE** specifies Serial Port D, where PC1 is RxD.

The **DOUTBUFSIZE** macro is specifies the size of the buffer used to send data to pin 3 of the XBee™ RF modem, which was configured for DIN. The **D** in **DOUTBUFSIZE** specifies Serial Port D, where PC0 is TxD.

The **SERD_RTS_PORT** macro identifies **PCDR**, the output flow control via Parallel Port C.

The **SERD_RTS_SHADOW** macro identifies the shadow register via Parallel Port C used by the output flow control.

The **SERD_RTS_BIT** macro identifies the bit on Parallel Port C, PC2, used by the output flow control.

The **SEND_CTS_PORT** macro identifies **PCDR**, the input flow control via Parallel Port C.

The **SERD_CTS_BIT** macro identifies the bit on Parallel Port C, PC3, used by the input flow control.

Serial Port C and D were selected for the sample programs since Serial Ports C and D are presented on J3, the RS-232 serial header on the RCM3720 Prototyping Board included with this Application Kit.

When interface the RF Interface module to an SF1000 socket via header J4, let's look at the macros for the RCM3300 Prototyping Board, which involves Serial Port D and Alternate Serial Port B.

```
#define ATCMDRSP_SP D   //set to serial port A, B, C, D, E, or F
#define DINBUFSIZE 511     // PC1 = RxD -- Xbee pin 2 = Dout
#define DOUTBUFSIZE 127    // PC0 = TxD -- Xbee pin 3 = Din
#define SERD_RTS_PORT PDDR // RTS is output flow control
#define SERD_RTS_SHADOW PDDRShadow
#define SERD_RTS_BIT 5     // PD5
#define SERD_CTS_PORT PDDR // CTS is input flow control
#define SERD_CTS_BIT 3     // PD3
#define DEFAULTBAUD 9600L  // xbee factory default baud rate
```

Function Reference Guide

The Dynamic C **Lib\zigbee\XBEE.LIB** library provides the function calls used with the ZigBee™/802/15.4 Application Kit. All Dynamic C implementations in support of the XBee™ RF modem are based on the AT command mode, and do not support the more proprietary API command mode.

xb_atModeOn

```
int xb_atModeOn(int guardTime);
```

DESCRIPTION

Puts the XBee™ RF modem in the AT command mode. The guard time must be expired before placing the modem in AT command mode. The default guard time is determined by the **xb_getGT()**—its default value is 0x3E8 (1000 ms), and the maximum is 3300 ms. To allow this function to honor the guard time, pass the guard time, otherwise pass 0.

NOTE: The guard time can be changed by calling **xb_setGT()**.

PARAMETER

guardTime the guard time; see **xb_setGT()** for more information. If the guard time is less than zero, the guard time is ignored.

RETURN VALUE

1 — success
-1 — error

xb_atModeOff

```
int xb_atModeOff();
```

DESCRIPTION

Explicitly exits the XBee™ RF modem from the AT command mode.

RETURN VALUE

1 — success
-1 — error

xb_setWR

```
int xb_setWR();
```

DESCRIPTION

This function call writes values to nonvolatile memory so that parameter modifications persist through subsequent resets.

NOTE: Once **`xb_setWR()`** is issued, no additional characters should be sent to the XBee™ RF modem until after the "OK\r" response is received.

RETURN VALUE

1 — success
-1 — error

xb_setRE

```
int xb_setRE();
```

DESCRIPTION

Restores the XBee™ RF modem parameters to their factory defaults.

RETURN VALUE

1 — success
-1 — error

xb_setFR

```
int xb_setFR();
```

DESCRIPTION

Resets the XBee™ RF modem. Responds immediately with an “OK,” then performs the reset ~100 ms later. Using the FR command will cause a network restart if either SC or ID was modified since the last reset.

RETURN VALUE

1 — success
-1 — error

xb_setNR

```
int xb_setNR(int nr);
```

DESCRIPTION

Resets the network layer parameters on one or more XBee™ RF modems within a personal area network. Responds immediately with an "OK," then causes a network restart. All network configuration and routing information is consequently lost. Can take up to 1.5 s for the XBee™ RF modem to reset the network, and when done the XBee™ RF modem will not be in the AT command mode.

If NR = 0, this function call resets the network layer parameters on the node issuing the command. This option is only supported on routers and end devices, and must be used with caution. Refer to the "*Resetting Coordinator*" section of the ***XBee™ ZigBee™ Protocol Manual*** for more information.

If NR = 1, this function call sends a broadcast transmission to reset the network layer parameters on all nodes in the personal area network. Refer to the "*Network Reset*" section of the ***XBee™ ZigBee™ Protocol Manual*** for more information.

PARAMETER

nr	0 – 1.

RETURN VALUE

1 — success
-1 — error

xb_getID

```
int xb_getID();
```

DESCRIPTION

Gets the personal area network PAN ID.

RETURN VALUE

0x0000 – 0x3FFF, 0xFFFF for a mesh network

0x0000 – 0xFFFF for a peer-to-peer network

xb_setID

```
int xb_setID(int id);
```

DESCRIPTION

Sets the personal area network PAN ID.

Coordinator — Set the preferred PAN ID (**`Set ID = 0xFFFF`**) to auto-select.

Router/End Device — Set the desired PAN ID. When the device searches for a coordinator, it attempts to only join to a parent that has a matching PAN ID. Set (**`ID = 0xFFFF`**) to join a parent operating on any PAN ID.

Changes to the PAN ID should be written to nonvolatile memory using the **`xb_setWR()`** function call. ID changes are not used until the XBee™ RF modem is reset (FR, NR or power-up).

PARAMETER

`id`	0x0000 – 0x3FFF, 0xFFFF for a mesh network 0x0000 – 0xFFFF for a peer-to-peer network

RETURN VALUE

1 — success
-1 — error

xb_getCH

```
int xb_getCH();
```

DESCRIPTION

Gets the channel number used for transmitting and receiving between RF modems. Uses peer-to-peer network protocol channel numbers.

.

0 = 0x0B	1 = 0x0C	2 = 0x0D	3 = 0x0E	4 = 0x0F
5 = 0x10	6 = 0x11	7 = 0x12	8 = 0x13	9 = 0x14
10 = 0x15	11 = 0x16	12 = 0x17	13 = 0x18	14 = 0x19
15 = 0x1A				

RETURN VALUE

0 – 15 (XBee)
1 – 12 (XBee-PRO)
-1 = error

xb_setCH

```
int xb_setCH(nt ch);
```

DESCRIPTION

Gets the channel number used for transmitting and receiving between XBee™ RF modems. Uses peer-to-peer network protocol channel numbers.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

PARAMETER

ch Possible values for **ch** (0–15 for XBee™, 1–12 for XBee™ PRO):

.

0 = 0x0B	1 = 0x0C	2 = 0x0D	3 = 0x0E	4 = 0x0F
5 = 0x10	6 = 0x11	7 = 0x12	8 = 0x13	9 = 0x14
10 = 0x15	11 = 0x16	12 = 0x17	13 = 0x18	14 = 0x19
15 = 0x1A				

RETURN VALUE

1 = success
-1 = error

xb_getDH

```
long xb_getDH();
```

DESCRIPTION

Destination Address High — Gets the upper 32 bits of the 64-bit destination address. This function call is not supported in the API command mode.

RETURN VALUE

0x00000000 – 0xFFFFFFFF

xb_setDH

```
int xb_setDH(long dh);
```

DESCRIPTION

Destination Address High — Sets the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the destination address used for transmission. 0x000000000000FFFF is the broadcast address for the personal area network. This function call is not supported in the API command mode.

PARAMETER

dh 0x00000000 – 0xFFFFFFFF

RETURN VALUE

1 = success
-1 = error

xb_getDL

```
long xb_getDL();
```

DESCRIPTION

Destination Address Low — Gets the lower 32 bits of the 64-bit destination address. This function call is not supported in the API command mode.

RETURN VALUE

0x00000000 – 0xFFFFFFFF

xb_setDL

```
int xb_setDL(long dl);
```

DESCRIPTION

Destination Address Low — Sets the lower 32 bits of the 64-bit destination address. When combined with DH, DL defines the destination address used for transmission. 0x000000000000FFFF is the broadcast address for the personal area network. This function call is not supported in the API command mode.

PARAMETER

dl	0x00000000 – 0xFFFFFFFF

RETURN VALUE

1 = success
-1 = error

xb_getMY

```
int xb_getMY();
```

DESCRIPTION

Gets the 16-bit source address of the XBee™ RF modem.

RETURN VALUE

0x0000 – 0xFFFF (16-bit address)

xb_setMY

```
int xb_setMY(int my);
```

DESCRIPTION

Sets the XBee™ RF modem's 16-bit source address.

Set MY = 0xFFFF to disable reception of packets with 16-bit addresses. The 64-bit source address (serial number) and broadcast address (0x000000000000FFFF) are always enabled.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

PARAMETER

my	0x0000 – 0xFFFF

RETURN VALUE

1 = success
-1 = error

xb_getSH

```
long xb_getSH();
```

DESCRIPTION

Reads the high 32 bits of the XBee™ RF modem's unique IEEE 64-bit address. The 64-bit source address is always enabled.

RETURN VALUE

0x00000000 – 0xFFFFFFFF

xb_getSL

```
long xb_getSL();
```

DESCRIPTION

Reads the low 32 bits of the XBee™ RF modem's unique IEEE 64-bit address. The 64-bit source address is always enabled.

RETURN VALUE

0x00000000 – 0xFFFFFFFF

xb_getRN

```
int xb_getRN();
```

DESCRIPTION

Gets the minimum value of the back-off exponent in the CSMA-CA algorithm (used for collision avoidance).

If RN = 0, collision avoidance is disabled during the first iteration of the algorithm (peer-to-peer network — **`macMinBE()`**).

Default: 0x0003.

RETURN VALUE

0x0000 – 0x0003
-1 = error

xb_setRN

```
int xb_setRN(int rn);
```

DESCRIPTION

Sets the minimum value of the back-off exponent in the CSMA-CA algorithm (used for collision avoidance).

If RN = 0, collision avoidance is disabled during the first iteration of the algorithm (peer-to-peer network — **`macMinBE()`**).

Default: 0x0003.

RETURN VALUE

1 = success
-1 = error

xb_getMM

```
int xb_getMM();
```

DESCRIPTION

Gets the MAC mode value.

Default: 0x0000.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

RETURN VALUE

0x0000 – 0x0002
-1 = error

xb_setMM

```
int xb_setMM(int mm);
```

DESCRIPTION

Sets the MAC mode value. The. MAC mode enables/disables the use of a MaxStream header in the peer-to-peer network RF packet.

When Mode 0 is enabled (MM = 0), duplicate packet detection and certain AT commands are enabled. Modes 1 and 2 are strictly peer-to-peer network modes.

Default: 0x0000.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

PARAMETER

mm	0 – 2

RETURN VALUE

1 = success
-1 = error

xb_getNI

```
int xb_getNI(char *ni);
```

DESCRIPTION

Reads back the node identifier.

PARAMETER

ni	pointer to a printable ASCII string up to 20 bytes

RETURN VALUE

1 = success
-1 = error

xb_setNI

```
int xb_setNI(char *ni);
```

DESCRIPTION

Stores a string identifier. The register only accepts printable ASCII data. A string can not start with a space in the AT command mode. A carriage return ends the command. The command will end automatically when the maximum bytes for the string have been entered. This string is returned as part of the **xb_setND()** (node discover) function call. This identifier is also used with the **xb_setDN()** (destination node) function call.

PARAMETER

ni	pointer to a printable ASCII string up to 20 bytes

RETURN VALUE

1 = success
-1 = error

xb_getND

```
int xb_getND(char *nd);
```

DESCRIPTION

Discovers and reports all XBee™ RF modems found.

Depending on the firmware that is loaded on the XBee™ RF modem, the following information is reported for each XBee™ RF modem discovered.

Peer-to-peer network firmware:

MY (my id)<CR>
SH (serial high)<CR>
SL (serial low)<CR>
DB (signal strength)<CR>
NI (node id)<CR> (variable length)

Mesh network firmware:

MY (my id)<CR>
SH (serial high)<CR>
SL (serial low)<CR>
NI (node id)<CR> (variable length)
PN (parent network)<CR> (2 bytes)
DT (device type)<CR> (1 byte: 0 = coordinator, 1 = router, 2 = end device)
ST (status)<CR> (1 byte: reserved)
PR (profile id)<CR> (2 bytes)
MI (manufacturer id)<CR> (2 bytes)
<CR>

The command ends after (NT * 100) milliseconds by returning a <CR>.

PARAMETER

nd pointer to a buffer that will contain all the nodes discovered.

NOTE: Make sure this buffer is big enough to hold all the nodes.

RETURN VALUE

1 = success
0 = pending
-1 = error

xb_getCE

```
int xb_getCE();
```

DESCRIPTION

Coordinator Enable — Sets/reads the coordinator setting.

Default: 0x0000.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

RETURN VALUE

0x0000 – 0x0001
-1 = error

xb_setCE

```
int xb_setCE(int ce);
```

DESCRIPTION

Coordinator Enable — Gets/reads the coordinator setting.

Default: 0x0000.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

PARAMETER

ce	0 – 1

RETURN VALUE

1 = success
-1 = error

xb_getNT

```
int xb_getNT();
```

DESCRIPTION

Node Discover Timeout — Gets the amount of time a node will spend discovering other nodes when **xb_getND()** or **xb_getDN()** is called.

Default: 0x3C (60d x 100 ms).

RETURN VALUE

0x0000 – 0x00FC
-1 = error

xb_setNT

```
int xb_setNT(int nt);
```

DESCRIPTION

Node Discover Timeout — Sets the amount of time a node will spend discovering other nodes when **xb_getND()** or **xb_getDN()** is called.

PARAMETER

rn	0x0000 – 0x00FC (rn x 100 m)

RETURN VALUE

1 = success
-1 = error

xb_getDN

```
int xb_getDN(char *dn);
```

DESCRIPTION

Destination Node — Resolves an node identifier (NI) string to a physical address (case sensitive). The following events occur after the destination node is discovered:

1. DL & DH are set to the module address with the matching NI parameter (AT command mode only).
2. The 64-bit and 16-bit addresses are sent out the UART.
3. The XBee™ RF modem automatically exits the AT command mode to allow immediate communication. If there is no response from a modem within (NT * 100) ms or a parameter is not specified (left blank), the command is terminated and an error message is returned. In the case of an error, the AT command mode is not exited.

PARAMETER

dn pointer to a printable ASCII string up to 20 bytes

RETURN VALUE

1 = success
-1 = error

xb_getSC

```
int xb_getSC();
```

DESCRIPTION

Gets the list of channels that are to be scanned.

Coordinator - Bit field list of channels to choose from prior to starting the network.

Router/End Device - Bit field list of channels that will be scanned to find a coordinator/router to join.

Bit (Channel):

.

0 = 0x0B	1 = 0x0C	2 = 0x0D	3 = 0x0E	4 = 0x0F
5 = 0x10	6 = 0x11	7 = 0x12	8 = 0x13	9 = 0x14
10 = 0x15	11 = 0x16	12 = 0x17	13 = 0x18	14 = 0x19
15 = 0x1A				

RETURN VALUE

0x0001 – 0xFFFF

xb_setSC

```
int xb_setSC(int sc);
```

DESCRIPTION

Sets the list of channels to scan.

Coordinator - Bit field list of channels to choose from prior to starting the network.

Router/End Device - Bit field list of channels that will be scanned to find a coordinator/router to join.

NOTE: Changes to **xb_setSC()** should be written using the **xb_setWR()** function call. **xb_setSC()** changes are not used until the XBee™ RF modem is reset (FR, NR or power-up).

Bit (Channel):

.

0 = 0x0B	1 = 0x0C	2 = 0x0D	3 = 0x0E	4 = 0x0F
5 = 0x10	6 = 0x11	7 = 0x12	8 = 0x13	9 = 0x14
10 = 0x15	11 = 0x16	12 = 0x17	13 = 0x18	14 = 0x19
15 = 0x1A				

PARAMETER

sc 0x0001 – 0xFFFF (bits 0, 13, 14, 15 are not allowed for the XBee-PRO™).

RETURN VALUE

1 = success
-1 = error

xb_getSD

```
int xb_getSD();
```

DESCRIPTION

Gets the scan duration exponent.

Coordinator — Duration of the active and energy scans (on each channel) that are used to determine an acceptable channel and the personal area network (PAN) ID for the coordinator to start up on.

Router / End Device — Duration of active scan (on each channel) used to locate an available coordinator/ router to join during association.

The scan time is measured as

(# Channels to Scan) * (2 ^ SD) * 15.36 ms

The number of channels to scan is determined by the SC parameter. The XBee™ RF modem can scan up to 16 channels (SC = 0xFFFF), and the XBee-PRO™ can scan up to 12 channels (0x1FFE).

Sample scan duration times (13-channel scan):

.

SD = 0	time = 0.200 s	SD = 2	time = 0.799 s
SD = 4	time = 3.190 s	SD = 6	time = 12.780 s
SD = 8	time = 51.120 s	SD = 10	time = 3.41 min
SD = 12	time = 13.63 min	SD = 14	time = 54.53 min

RETURN VALUE

0x0000 – 0x0007
-1 = error

xb_setSD

```
int xb_setSD(int sd);
```

DESCRIPTION

Gets the scan duration exponent. Changes to SD should be written using WR command.

Coordinator — Duration of the active and energy scans (on each channel) that are used to determine an acceptable channel and the personal area network (PAN) ID for the coordinator to start up on.

Router / End Device — Duration of active scan (on each channel) used to locate an available coordinator/ router to join during association.

The scan time is measured as

(# Channels to Scan) * (2 ^ SD) * 15.36 ms

The number of channels to scan is determined by the SC parameter. The XBee™ RF modem can scan up to 16 channels (SC = 0xFFFF), and the XBee-PRO™ can scan up to 12 channels (0x1FFE).

Sample scan duration times (13-channel scan):

.

SD = 0	time = 0.200 s	SD = 2	time = 0.799 s
SD = 4	time = 3.190 s	SD = 6	time = 12.780 s
SD = 8	time = 51.120 s	SD = 10	time = 3.41 min
SD = 12 (default)	time = 13.63 min	SD = 14	time = 54.53 min

PARAMETER

sd 0x0000 – 0x0007

RETURN VALUE

1 = success
-1 = error

xb_getA1

```
int xb_getA1();
```

DESCRIPTION

End Device Association — Gets/reads the association options for the end device.

Default: 0x0000.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

RETURN VALUE

0x0000 – 0x000F
-1 = error

xb_setA1

```
int xb_setA1(int a1);
```

DESCRIPTION

End Device Association — Sets/reads the association options for the end device.

Bit 0 — reassign personal area network (PAN) ID
- 0 — will only associate with a coordinator operating on a PAN ID that matches the modem's ID
- 1 — may associate with a coordinator operating on any PAN ID

Bit 1 — reassign channel
- 0 — will only associate with a coordinator operating on a channel that matches the CH setting
- 1 — may associate with a coordinator operating on any channel

Bit 2 — auto-associate
- 0 — device will not attempt association
- 1—-device attempts association until success

NOTE: This association is only supported in peer-to-peer networks. End devices must always associate with a coordinator.

Bit 3 — poll coordinator on pin wake
- 0 — pin wake will not poll the coordinator for indirect (pending) data
- 1 — pin wake will send poll request to the coordinator to extract any pending data

Bits 4–7 are reserved.

Default: 0x0000.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

PARAMETER

a1	0x00 – 0x0F (bit field)

RETURN VALUE

1 = success
-1 = error

xb_getA2

```
int xb_getA2();
```

DESCRIPTION

Gets the coordinator association options.

Default: 0x0007.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

RETURN VALUE

0x0000 – 0x0007
-1 = error

xb_setA2

```
int xb_setA2(int a2);
```

DESCRIPTION

Sets the coordinator association options.

Bit 0 — reassign personal area network (PAN) ID

0 — coordinator will not perform an active scan to locate available PAN IDs. It will operate on ID (PAN ID).

1 — coordinator will perform an active scan to locate an available PAN ID. If a PAN ID conflict is found, the ID parameter will change.

Bit 1 — reassign channel

0 — coordinator will not perform an energy scan to locate a free channel. It will operate on the channel determined by the CH parameter.

1 — coordinator will perform an energy scan to find a free channel, and will then operate on that channel.

Bit 2 — allow association

0 — coordinator will not allow any devices to associate to it.

1— coordinator will allow devices to associate to it.

Bits 3–7 are reserved

Default: 0x0006.

NOTE: This function call is only supported in peer-to-peer networks, and is ***not*** supported with mesh networks.

PARAMETER

a2	0x00 – 0x07 (bit field)

RETURN VALUE

1 = success

-1 = error

xb_getNJ

```
int xb_getNJ();
```

DESCRIPTION

Gets the time that a coordinator/router allows nodes to join.

RETURN VALUE

0x0000 – 0x00FF (default is 0xFF to always join)
-1 = error

xb_setNJ

```
int xb_setNJ(int nj);
```

DESCRIPTION

Sets the time that a coordinator/router allows nodes to join. This value can be changed at run time without requiring a coordinator or router to restart.The time starts once the coordinator or router has started. The timer is reset on a power cycle or when NJ changes.

Default: 0xFF always allows joins.

PARAMETER

nj	0x0000 – 0x00FF (x 100 ms)

RETURN VALUE

1 = success
-1 = error

xb_getAI

```
int xb_getAI();
```

DESCRIPTION

Association Indication — Reads information regarding the join request for the last node.

RETURN VALUE

0x00 = successful completion — coordinator started or router/end device found and joined with a parent

0x21= scan found no personal area networks

0x22 = scan found no valid personal area networks based on current SC and ID settings

0x23 = valid coordinator or routers found, but they are not allowing joining (NJ expired)

0x24 = router full, allows join set, but cannot allow any more routers to join

0x25 = router full, allows join set, but cannot allow any more end devices to join

0x26 = cannot join to a node because it was a child or descendent of this device

0x27 = node joining attempt failed

0x28 = device is orphaned and is looking for its parent using orphan scans

0x29 = router start attempt failed

0x2A = coordinator start attempt failed

0xFF = scanning for a parent

-1 = error

xb_getPL

```
int xb_getPL();
```

DESCRIPTION

Gets the power level at which the XBee™ RF modem transmits conducted power.

RETURN VALUE

0x0000 – 0x0004 (power level)
-1 = error

xb_setPL

```
int xb_setPL(int pl);
```

DESCRIPTION

Sets the power level at which the XBee™ RF modem transmits conducted power.

PARAMETER

pl	0x0000 = -10 dBm 0x0001 = - 6 dBm 0x0002 = - 4 dBm 0x0003 = - 2 dBm 0x0004 = 0 dBm (default)

NOTE: Power levels will be different for the XBee-PRO™.

RETURN VALUE

1 = success
-1 = error

xb_getCA

```
int xb_getCA();
```

DESCRIPTION

Gets the CCA (clear channel assessment) threshold. Prior to transmitting a packet, a CCA is performed to detect the energy on the channel. If the detected energy is above the CCA threshold, the XBee™ RF modem will not transmit the packet.

Default: 0x0040 (-64 dBm).

RETURN VALUE

0x0000 – 0x0050 [-dBm]
-1 = error

xb_setCA

```
int xb_setCA(int ca);
```

DESCRIPTION

Sets the CCA (clear channel assessment) threshold. Prior to transmitting a packet, a CCA is performed to detect the energy on the channel. If the detected energy is above the CCA threshold, the XBee™ RF modem will not transmit the packet.

Default: 0x0040 (-64 dBm).

PARAMETER

ca	0x0000 – 0x0050 [-dBm]

RETURN VALUE

1 = success
-1 = error

xb_setAP

```
int xb_setAP(int ap);
```

DESCRIPTION

Enables the API mode. When **ap** = 1 or 2, the API mode is only supported by XBee™ RF modems that contain the following firmware versions:

- 8.1xx (coordinator)
- 8.3xx (router)
- 8.5xx (end device)

Default: 1 (API).

PARAMETER

ap	0x0001 = API-enabled 0x0002 = API-enabled (with escape control characters)

RETURN VALUE

1 = success
-1 = error

xb_getBD

```
int xb_getBD();
```

DESCRIPTION

Gets the serial interface data rate for communications between the XBee™ RF modem's serial port and the Rabbit microprocessor.

Default: 0x03 (9600 bps).

RETURN VALUE

0x00 – 0x07 (standard baud rates)
-1 = error

xb_setBD

```
int xb_setBD(int bd);
```

DESCRIPTION

Sets the serial interface data rate for communications between the XBee™ RF modem's serial port and the Rabbit microprocessor.

0 – 7 (standard baud rates):
- 0 = 1200 bps
- 1 = 2400 bps
- 2 = 4800 bps
- 3 = 9600 bps
- 4 = 19200 bps
- 5 = 38400 bps
- 6 = 57600 bps
- 7 = 115200 bps

PARAMETER

bd 0x00 – 0x07

RETURN VALUE

1 = success
1 = error

xb_getNB

```
int xb_getNB();
```

DESCRIPTION

Gets the serial parity settings.

RETURN VALUE

0 = no parity
1 = even parity
2 = odd parity
3 = mark
4 = space
-1 = error

xb_setNB

```
int xb_setNB(int nb);
```

DESCRIPTION

Sets the serial parity settings:

0 = no parity
1 = even parity
2 = odd parity
3 = mark
4 = space

PARAMETER

nb 0x00 – 0x04

RETURN VALUE

1 = success
-1 = error

xb_getRO

```
int xb_getRO();
```

DESCRIPTION

Packetization Timeout — Gets the number of character times of inter-character silence required before packetization.

RETURN VALUE

0x0000 – 0x00FF [x character times]
-1 = error

xb_setRO

```
int xb_setRO(int ro);
```

DESCRIPTION

Packetization Timeout — Sets the number of character times of inter-character silence required before packetization. Set RO = 0 to transmit characters as they arrive instead of buffering them into one RF packet.

Default: 3.

PARAMETER

ro	0x00 – 0xFF (x character times)

RETURN VALUE

1 = success
-1 = error

xb_getD7

```
int xb_getD7();
```

DESCRIPTION

DIO7 Configuration — Gets the options for the DIO7 line of the XBee™ RF modem.

RETURN VALUE

0 = disabled
1 = CTS flow control
-1 = error

xb_setD7

```
int xb_setD7(int d7);
```

DESCRIPTION

DIO7 Configuration — Selects/reads the options for the DIO7 line of the XBee™ RF modem.

Default: 1.

PARAMETER

d7	0 = disabled 1 = CTS flow control

RETURN VALUE

1 = success
-1 = error

xb_getD5

```
int xb_getD5();
```

DESCRIPTION

DIO5 Configuration — Configures the options for the DIO5 line of the XBee™ RF modem. Options include: associated LED indicator (LED blinks when associated).

RETURN VALUE

0 = disabled
1 = associated LED indicator
-1 = error

xb_setD5

```
int xb_setD5(int d5);
```

DESCRIPTION

DIO5 Configuration — Configures the options for the DIO5 line of the XBee™ RF modem. Options include: associated LED indicator (LED blinks when associated).

Default: 1.

PARAMETER

d5	0 = disabled 1 = associated LED indicator

RETURN VALUE

1 = success
-1 = error

xb_getD4

```
int xb_getD4();
```

DESCRIPTION

DIO4 Configuration — Gets the options for the DIO4 line of the XBee™ RF modem.

RETURN VALUE

0 = disabled
1 = N/A
2 = ADC
3 = D
4 = DO low
5 = DO high
-1 = error

xb_setD4

```
int xb_setD4(int d4);
```

DESCRIPTION

DIO4 Configuration — Selects/reads the options for the DIO4 DIO4 line of the XBee™ RF modem.

Default: 0 Disabled.

PARAMETER

d4

0 = disabled
1 =N/A
2 = ADC
3 = DI
4 = DO low
5 = DO high

RETURN VALUE

1 = success
-1 = error

xb_getD3

```
int xb_getD3();
```

DESCRIPTION

DIO3 Configuration — Gets the options for the DIO3 line of the XBee™ RF modem.

RETURN VALUE

0 = disabled
1 = N/A
2 = ADC
3 = DI
4 = DO low
5 = DO high
-1 = error

xb_setD3

```
int xb_setD3(int d3);
```

DESCRIPTION

DIO3 Configuration — Selects/reads the options for the DIO3 line of the XBee™ RF modem.

Default: 0 (disabled).

PARAMETER

d3	0 = disabled 1 =N/A 2 = ADC 3 = DI 4 = DO low 5 = DO high

RETURN VALUE

1 = success
-1 = error

xb_getD2

```
int xb_getD2();
```

DESCRIPTION

DIO2 Configuration — Gets the options for the DIO2 line of the XBee™ RF modem.

RETURN VALUE

0 = disabled
1 = N/A
2 = ADC
3 = DI
4 = DO low
5 = DO high
-1 = error

xb_setD2

```
int xb_setD2(int d2);
```

DESCRIPTION

DIO2 Configuration — Selects/reads the options for the DIO2 line of the XBee™ RF modem.

Default: 0 (disabled).

PARAMETER

d2	0 = disabled 1 =N/A 2 = ADC 3 = DI 4 = DO low 5 = DO high

RETURN VALUE

1 = success
-1 = error

xb_getD1

```
int xb_getD1();
```

DESCRIPTION

DIO1 Configuration — Gets the options for the DIO1 line of the XBee™ RF modem.

RETURN VALUE

0 = disabled
1 = N/A
2 = ADC
3 = DI
4 = DO low
5 = DO high
-1 = error

xb_setD1

```
int xb_setD1(int d1);
```

DESCRIPTION

DIO1 Configuration — Selects/reads the options for the DIO1 line of the XBee™ RF modem.

Default: 0 (disabled).

PARAMETER

d1	0 = disabled 1 =N/A 2 = ADC 3 = DI 4 = DO low 5 = DO high

RETURN VALUE

1 = success
-1 = error

xb_getD0

```
int xb_getD0();
```

DESCRIPTION

DIO0 Configuration — Gets the options for the DIO0 line of the XBee™ RF modem.

RETURN VALUE

0 = disabled
1 = N/A
2 = ADC
3 = DI
4 = DO low
5 = DO high
-1 = error

xb_setD0

```
int xb_setD0(int d0);
```

DESCRIPTION

DIO0 Configuration — Selects/reads the options for the DIO0 line of the XBee™ RF modem.

Default: 0 (disabled).

PARAMETER

d1	0 = disabled 1 =N/A 2 = ADC 3 = DI 4 = DO low 5 = DO high

RETURN VALUE

1 = success
-1 = error

xb_setIU

```
int xb_setIU(int iu);
```

DESCRIPTION

Enables the I/O data received wirelessly to be sent out UART. The data are sent using an API frame regardless of the current ATAP mode.

Default: 1.

PARAMETER

iu	0 = disabled 1 = enabled

RETURN VALUE

1 = success
-1 = error

xb_getIS

```
int xb_getIS();
```

DESCRIPTION

Forces a read of all the local XBee™ RF modems' enabled inputs (DI or ADC).

Sample data:

```
<#of samples><channel indicator><active dios><
<byte>\r<byte>\r<byte>\r<byte>\r
```

RETURN VALUE

1 = success
-1 = error

xb_getP0

```
int xb_getP0();
```

DESCRIPTION

PWM0 Configuration — Gets function for PWM0.

RETURN VALUE

0 = disabled
1 = RSSI PWM
-1 = error

xb_setP0

```
int xb_setP0(int p0);
```

DESCRIPTION

PWM0 Configuration — Sets function for PWM0.

Default: 1.

PARAMETER

p0	0 = disabled 1 = RSSI PWM

RETURN VALUE

1 = success
-1 = error

xb_getRP

```
int xb_getRP();
```

DESCRIPTION

RSSI PWM Timer — Gets time RSSI signal will be output after last transmission. When RP = 0, output will always be on.

RETURN VALUE

0x0000 - 0x00FF (x 100 ms)
-1 = error

xb_setRP

```
int xb_setRP(int rp);
```

DESCRIPTION

RSSI PWM Timer — Sets time RSSI signal will be output after last transmission. When RP = 0, output will always be on.

Default: 0x28.

PARAMETER

rp	0x0000 – 0x00FF (x 100 ms)

RETURN VALUE

1 = success
-1 = error

xb_getVR

```
int xb_getVR();
```

DESCRIPTION

Reads the firmware version of the XBee™ RF modem.

Default: factory set.

RETURN VALUE

0x0000 – 0xFFFF [read-only]

xb_getHV

```
int xb_getHV();
```

DESCRIPTION

Reads the hardware version of the XBee™ RF modem.

Default: factory set.

RETURN VALUE

0x0000 – 0xFFFF [read-only]

xb_getCT

```
int xb_getCT();
```

DESCRIPTION

Command Mode Timeout — Gets the period of inactivity (no valid commands received) after which the XBee™ RF modem automatically exits AT command mode and returns to the idle mode.

Default: 0x64.

RETURN VALUE

0x0002 – 0x028F (x 100 ms)
-1 = error

xb_setCT

```
int xb_setCT(int gt);
```

DESCRIPTION

Command Mode Timeout — Sets the period of inactivity (no valid commands received) after which the XBee™ RF modem automatically exits AT command mode and returns to the idle mode.

Default: 0x64.

PARAMETER

gt	0x0002 – 0x028F (x 100 ms)

RETURN VALUE

1 = success
-1 = error

xb_getGT

```
int xb_getGT();
```

DESCRIPTION

Guard Times — Gets the required period of silence before and after the command sequence characters of the AT command mode sequence (GT + CC + GT). The period of silence is used to prevent inadvertent entrance into AT command mode.

Default: 0x03E8.

RETURN VALUE

0x0001 – 0x0CE4 (x 1 ms, max of 3.3 decimal seconds)
-1 = error

xb_setGT

```
int xb_setGT(int gt);
```

DESCRIPTION

Guard Times — Sets the required period of silence before and after the command sequence characters of the AT command mode sequence (GT + CC + GT). The period of silence is used to prevent inadvertent entrance into AT command mode.

Default: 0x03E8.

PARAMETER

gt	0x0001 – 0x0CE4 (x 1 ms, max of 3.3 decimal seconds)

RETURN VALUE

1 = success
-1 = error

xb_getCC

```
int xb_getCC();
```

DESCRIPTION

Command Sequence Character — Gets the ASCII character value to be used between guard times of the AT command mode sequence (GT + CC + GT). The AT command mode sequence enters the XBee™ RF modem into the AT command mode. The CC command is only applicable when using modems that contain the following “AT command” mesh network firmware versions:

- 8.0xx (coordinator)
- 8.2xx (router),
- 8.4xx (end device)

Default: 0x2B ('+' ASCII)

RETURN VALUE

0x0000 – 0x00FF
-1 = error

xb_setCC

```
int xb_setCC(int cc);
```

DESCRIPTION

Command Sequence Character — Sets the ASCII character value to be used between guard times of the AT command mode sequence (GT + CC + GT). The AT command mode sequence enters the XBee™ RF modem into the AT command mode. The CC command is only applicable when using modems that contain the following “AT command” mesh network firmware versions:

- 8.0xx (coordinator)
- 8.2xx (router),
- 8.4xx (end device)

Default: 0x2B ('+' ASCII)

PARAMETER

cc	0x0000 – 0x00FF

RETURN VALUE

1 = success
-1 = error

Specifications

Table A-1. ZigBee™ Application Kit RF Interface Module Specifications

Features		RF Interface Module
RF Module		MaxStream XBee™
Compliance		802.15.4 standard (ZigBee™ compliant)
Frequency		ISM 2.4 GHz
Performance	Indoor Range	100 ft (30 m)
	Outdoor Line-of-Sight Range	300 ft (90 m)
	Transmit Power Output	1 mW (0 dBm)
	RF Data Rate	250,000 bps
	Receiver Sensitivity	-92 dBm (1% PER)
Antenna		Chip antenna
Supported Network Topologies		• Point-to-point • Point-to-multipoint • Peer-to-peer • Mesh
Number of RF Channels		16 direct-sequence channels (software-selectable)
Filtration Options		• PAN ID • Channel • Source/destination addresses
Power (typical)	Transmit	55 mA @ 3.5–6.0 V
	Idle/Receive	60 mA @ 3.5–6.0 V
Battery Pack		3 AAA, 540 mA•h to 0.8 V each battery
Operating Temperature		–40°C to +70°C
Humidity		5% to 95%, noncondensing
Connectors		Two 2 × 10, 0.1" pitch sockets One power connector One 2 × 5, 2 mm pitch serial header One 2 × 5, 0.1" pitch serial header
Board Size with XBee™ RF Modem Installed (XBee™ RF modem extends 0.2" [5.1 mm] beyond edge of board)		2.00" × 2.00" × 0.50" (51 mm × 51 mm × 13 mm)

Complete specifications for the RCM3720 RabbitCore module and the RCM3720 Prototyping Board are available in the ***RCM3700 User's Manual***.

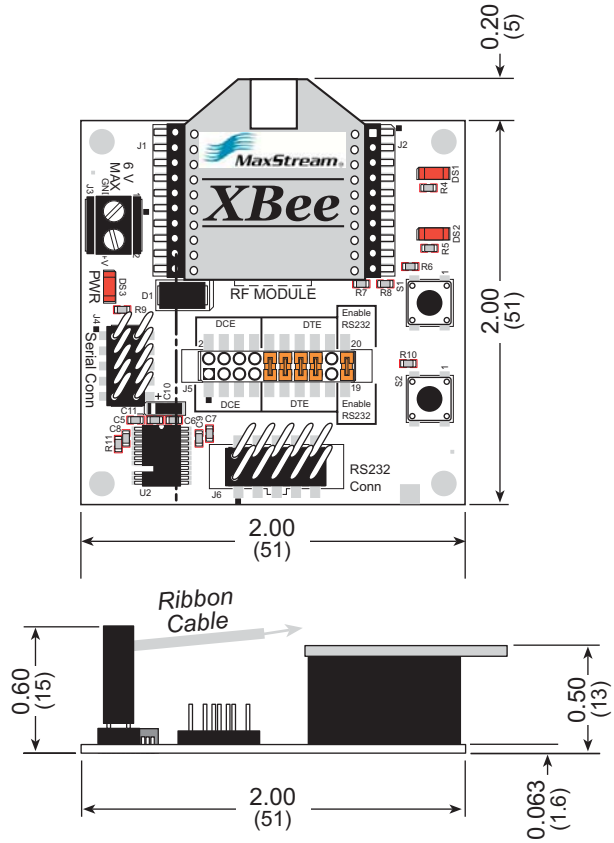


Figure A-1. RF Interface Module Dimensions
(with XBee™ RF Modem and Serial Ribbon Cable Installed)

NOTE: All measurements are in inches followed by millimeters enclosed in parentheses. All dimensions have a manufacturing tolerance of ±0.01" (0.25 mm).