Millennial Net

Wireless Sensor Networking…Anywhere

EK-5209-5 Evaluation Kit

i-Bean® 916 MHz Wireless Sensor Network

User's Guide

Document Number: DOC-0005
Revision: 01
Released: July 2004

COPYRIGHT

NOTICE

Millennial Net, Inc.
2 4th Avenue
Burlington, MA 01803-3304 USA
781.222.1030

CAUTION

All device installation, configuration, and reconfiguration must be performed only by qualified service personnel.

Initialization of the product should be performed only by a qualified systems administrator.

Compliance Statement

FCC compliance for Millennial Net's EK-5209-5 Evaluation Kit (916MHz, 5-3-1) consisting of the following models/components:

- iB-5209 Endpoint
- GW-5209 Gateway
- RT-5209 Router

Compliance Statement (Part 15.19)

The Millennial Net EK-5209-5 Evaluation Kit complies with Part 15 of the FCC Rules and with RSS-210 of Industry Canada.

Operation is subject to the following two conditions:

(1) This device may not cause harmful interference, and

(2) This device must accept any interference received, including interference that may cause undesired operation.

Warning (Part 15.21)

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment

Trademarks

Information subject to change.

Table of Contents

About This Guide

Audience xiv
How to Use This Guide xiv
Symbols and Conventions xv
Contacting Millennial Net xvi
World Wide Web xvi
Customer Support xvi
Technical Publications xvi

1 Introduction

Product Overview 1-2
Major Features 1-2
i-Bean Network Overview 1-3
Evaluation Kit Contents 1-5
Host PC Requirements 1-5

2 Kit Installation

Installing the i-Bean Wireless Network 2-2
Hardware Installation 2-3
i-Bean Gateway Setup (GW-5209) 2-3
i-Bean Router Setup (RT-5209) 2-6
i-Bean Endpoint Setup (iB-5209) 2-9
iB-5209 Network Monitor Installation 2-10
Installing contents of Millennial Net's iB-5209 CD-ROM 2-10
Launching iB-5209 Network Monitor 2-12

3 iB-5209 Network Monitor Operations

iB-5209 Network Monitor Overview 3-2
Menu Bar 3-3
Gateway 3-4
Device Counts 3-4
Node Details 3-4
Setting Thread Priority 3-6
Configuring a Node's Operation 3-7
Using Watch function to display current I/O information 3-9
Configuring Sample Interval of Single Node 3-11
Configuring Sample Interval of all Network Nodes 3-11
Configuring Digital I/O Operation 3-12
Configuring UART Operation 3-15
Configuring AD (analog-to-digital) Converter Operation 3-17
Configuring RS-232 Operation (RT-5209 only) 3-18
Configuring RS-485 Operation (RT-5209 only) 3-19
Labeling i-Bean Endpoint or i-Bean Router 3-20
Creating an Event Log File 3-21

Configure Persistence Attributes 3-22
Configure Serial and ADC Data Formats 3-23
Select Com Port on Host PC 3-24
View Monitor Statistics 3-25
View Contents of Event Log File 3-26
Enable Multiple Capture 3-28

A API Functions

iB-5209 API Overview A-2
i-Bean API Functions Overview A-4
iBeanAPI.h A-5
    Data Structures A-5
    Functions A-10
iBeanAPI_IO.h A-16
    Data Structures A-16
    Functions A-17
iBeanAPI_Utils.h A-24
    Functions A-24
Example API Code A-27
    ListDevicesVC7 Example A-27
    ListDevicesVC7 Code A-28

B Sample Application

Application Overview B-2
    Application Components B-3
Application Setup & Operation B-4
    Temperature Sensor Assembly Setup B-4
    Launching TempMonitor Application B-5
    TempMonitor Overview B-6
Changing Temperature Sensor Battery B-7

Index

List of Figures

Figure 1-1. i-Bean network 1-3
Figure 1-2. Sample i-Bean network topologies 1-4
Figure 2-1. i-Bean Gateway components (external) 2-3
Figure 2-2. Powering the i-Bean Gateway with batteries 2-4
Figure 2-3. i-Bean Router components 2-7
Figure 2-4. i-Bean Endpoint and terminal board (top and bottom views) 2-9
Figure 2-5. Installing Millennial Net's CD-ROM contents 2-11
Figure 2-6. Launching iB-5209 Network Monitor 2-12
Figure 3-1. Sample iB-5209 Network Monitor window 3-2
Figure 3-2. iB-5209 Network Monitor's Edit Device window 3-7
Figure 3-3. Displaying I/O information using Watch function 3-9
Figure 3-4. Configuring sample interval of single node 3-11
Figure 3-5. Configuring sample interval of all nodes 3-12
Figure 3-6. Configuring i-Bean Endpoint/terminal board for digital I/O 3-14
Figure 3-7. Configuring i-Bean Endpoint/i-Bean Router for UART 3-16
Figure 3-8. Configuring i-Bean Endpoint/i-Bean Router for analog I/O 3-17
Figure 3-9. Configuring i-Bean Router for RS-232 3-18
Figure 3-10. Configuring i-Bean Router for RS-485 3-19
Figure 3-11. Labeling i-Bean Endpoint or i-Bean Router 3-20
Figure 3-12. Configure an event log file 3-21
Figure 3-13. Configuring node persistence attributes 3-22
Figure 3-14. Configuring display attributes of iB-5209 Network Monitor 3-23
Figure 3-15. Selecting com port on host PC 3-24
Figure 3-16. Viewing monitored statistics 3-25
Figure 3-17. View contents of event log file 3-26
Figure 3-18. Enable Multiple Capture 3-28
Figure A-1. iBeanAPI directories A-2
Figure A-2. API Example: ListDevicesVC7 ..... A-27
Figure B-1. Temperature sensor assembly overview (cover removed) B-2
Figure B-2. Process flow B-3
Figure B-3. Installing i-Bean Endpoint in Temperature Sensor Assembly B-4
Figure B-4. Launching TempMonitor B-5
Figure B-5. TempMonitor display B-6
Figure B-6. Changing temperature sensor assembly battery B-7

List of Tables

Table 2-1. i-Bean Gateway Status LEDs 2-5
Table 2-2. i-Bean Router Status LEDs..... 2-8
Table 3-1. Edit Device window functions..... 3-7
Table 3-2. Watch window functions 3-9
Table 3-3. Event Log Key Definitions 3-26
Table A-1. i-Bean API functions A-4

About This Guide

This section provides information related to the content of the user guide:

- 'Audience' on page -xiv
- 'How to Use This Guide' on page -xiv
- 'Symbols and Conventions' on page -xv
- 'Contacting Millennial Net' on page -xvi

Audience

This guide is intended for the following qualified service personnel who are responsible for installing and operating the EK-5209-5 Evaluation Kit:

- System installer
- Hardware technician
- System operator
- System administrator

How to Use This Guide

The sections of this guide provide the following information:

Section	Provides
Chapter 1, "Introduction"	Overview of i-Bean network.
Chapter 2, "Kit Installation"	Instructions for installing the hardware (i-Bean Gateway, i-Bean Routers, i-Bean Endpoints) and iB-5209 Network Monitor (GUI).
Chapter 3, "iB-5209 Network Monitor Operations"	Procedures for using iB-5209 Network Monitor software to configure the i-Bean network nodes. Also includes information for attaching external I/O devices to an i-Bean Endpoint or i-Bean Router.
Appendix A, "API Functions"	Information on the iB-5209 API functions.
Appendix B, "Sample Application"	Procedure for running the sample application provided with the evaluation kit.
Index	An alphabetical index of topics described in this manual.

Symbols and Conventions

This guide uses the following symbols and conventions to emphasize certain information.

Note: A note is used to highlight important information relating to the topic being discussed.

Caution

A caution means that a specific action could cause harm to the equipment or to the data.

Warning

A warning describes an action that could result in physical injury, or destruction of property.

Hazard

A hazard is a particular form of warning related expressly to electric shock.

Italics - Indicate the first occurrence of a new term, book title, and emphasized text.

1. Numbered list - Where the order of the items is important.

- Bulleted list - Where the items are of equal importance and their order is unimportant.

Contacting Millennial Net

World Wide Web

Millennial Net maintains a site on the World Wide Web where information on the company and its products can be found. The URL is:

www.millennialnet.com

Customer Support

For answers to your technical questions, Millennial Net's Customer Service department can be reached at:

phone:
781.222.1030

e-mail:
support@millennialnet.com

Technical Publications

Millennial Net is committed to providing you with quality technical documentation. Your feedback is valuable and appreciated. Please send comments, suggestions, and enhancements regarding this guide or any Millennial Net documentation to:

support@millennialnet.com

Please include the document title, number, and version in your email.

1

Introduction

This chapter provides an overview of the i-Bean network and evaluation kit. In this chapter you will find:

- 'Product Overview' on page 1-2
- 'i-Bean Network Overview' on page 1-3
- 'Evaluation Kit Contents' on page 1-5
- 'Host PC Requirements' on page 1-5

Product Overview

Millennial Net's EK-5209-5 Evaluation Kit contains everything you need to set up a self-organizing, wireless star-mesh network. Once installed, you are able to observe the performance and operation of the network components and prototype your application.

The EK-5209-5 Evaluation Kit contains one i-Bean Gateway, three i-Bean Routers, and five i-Bean Endpoints. Software included with the kit includes the Application Program Interface (API) and iB-5209 Network Monitor, which is the network monitoring tool and graphic user interface (GUI). Also included, is a temperature sensor assembly for running a sample iB-5209 application (see Appendix B, "Sample Application").

Documentation for the evaluation kit includes this user's guide, which describes how to set up the evaluation network, including connections to the host computer, power supplies, sensors, and other devices. For complete details on the contents of the evaluation kit, refer to 'Evaluation Kit Contents' on page 1-5.

iB-5209 Network Monitor and API software run on MS Windows XP and 2000. iB-5209 Network Monitor allows you to set network and device operating parameters and monitor the status of the network components and their inputs/outputs. The API software can be easily incorporated into user application programs written in Microsoft Visual C/C++.

Major Features

Major features of the EK-5209-5 Evaluation Kit include the following:

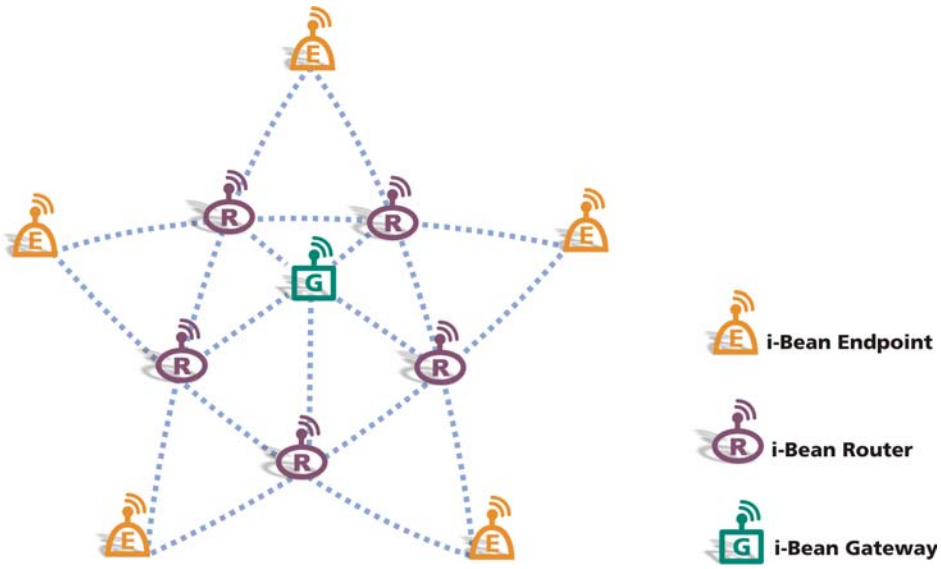
- Frequency band: 916 MHz
- Bi-directional/multiple-access communication
- Under ideal environmental conditions, 60 feet (20 m) line-of-sight transmission distance from i-Bean Endpoints to i-Bean Router or i-Bean Gateway; 90 feet (30 m) between i-Bean Router and i-Bean Gateway.
- iB-5209 Network Monitor graphic user interface (GUI) for configuring the i-Bean network and evaluating its performance.
- Application Programming Interface (API).
- i-Bean Endpoint and i-Bean Router-specific features include:
  - Configurable sampling interval
  - Digital I/O - 4 channels
  - ADC input - 4 channels
  - UART output

i-Bean Network Overview

Millennial Net's innovative, self-organizing network technology combines micro-power sensor interface i-Bean Endpoints and i-Bean Routers with an i-Bean Gateway to form a reliable, scalable star-mesh wireless network (see Figure 1-1). This is a unique solution for low data-rate networks that provides both long battery life at the sensor interfaces and fault-tolerant networking. Our patent-pending network protocol creates robust, fully redundant wireless links from the i-Bean Gateway to the i-Bean Endpoints through a self-configuring mesh network. The i-Bean Endpoints directly connect to, and can fit inside, analog or digital sensors and actuators.

Each network device is configured at the factory with a unique *device ID* and a *group ID*. The device ID indentifies the device within a network, while the group ID identifies the i-Bean network that the device is associated with. Both IDs are statically assigned and cannot be changed by a system user. The group ID allows i-Bean devices to establish networks within the same location without interfering with each other. Devices (i-Bean Endpoints and i-Bean Routers) can join the network only if they have the same group ID that is assigned to the i-Bean Gateway. The network elements self-organize at power-up and re-configure in response to changes in the environment, network traffic, device status, and location. These tiny devices enable mobility and minimize installation and operating costs.

Figure 1-1. i-Bean network



i-Bean Endpoints contain a microcomputer, device I/O, and a wireless transceiver. They interface to analog and/or digital (serial or parallel) sensors and actuators. The tiny endpoints (about 38 x 15 mm including the antenna) can fit inside sensors and actuators and run on small, low-cost batteries for years.

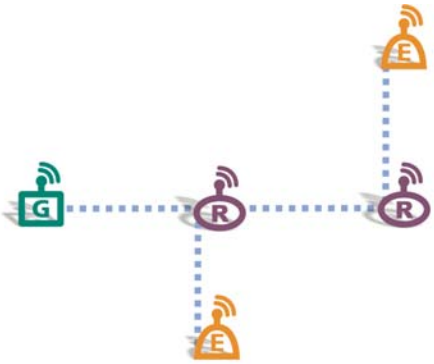
i-Bean Routers extend network coverage area, route around obstacles, and provide back-up routes in case of network congestion or device failure. Like i-Bean Endpoints, i-Bean Routers may also be connected via analog and digital interfaces to sensors and actuators.

i-Bean Gateways interface the i-Bean network to your host PC. They aggregate data from the network and act as a portal to monitor performance and configure network parameters. i-Bean Gateways connect via an RS-232 connection to the host PC, which in turn could be connected to a LAN or the Internet through one or more physical-layer interfaces, including RS-232, Ethernet, 802.11, landline or cellular modem.

i-Bean Endpoints, i-Bean Routers, and i-Bean Gateways self-organize at power-up and quickly re-configure as devices join, leave, or move around the network. They also adapt to changes in network traffic and propagation conditions. These capabilities enable mobility of individual devices or the entire network, and minimize installation and operating costs.

Millennial Net's protocol and components support several topologies including the samples illustrated in Figure 1-2.

Figure 1-2. Sample i-Bean network topologies

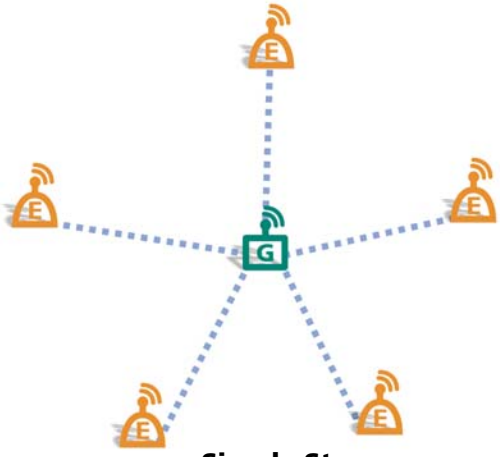


Linear

Consisting of single path between i-Bean Gateway, i-Bean Routers, and i-Bean Endpoints

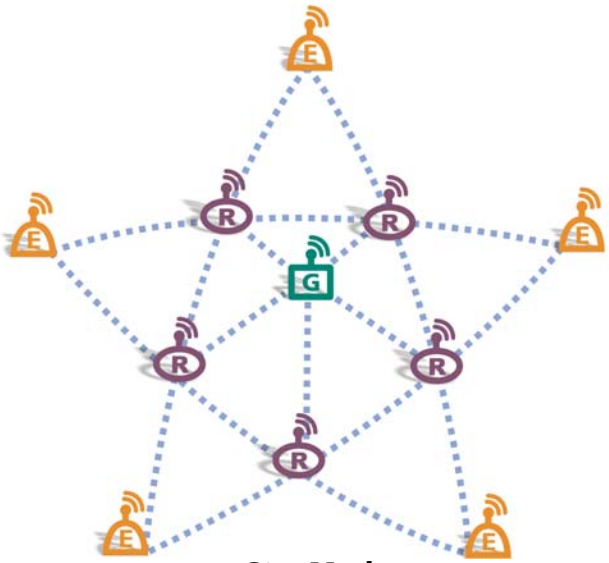
Simple Mesh

Consisting of homogeneous i-Bean Routers with an i-Bean Gateway



Simple Star

Combining low power i-Bean Endpoints with an i-Bean Gateway



Star Mesh

Combining reliability and low power

Evaluation Kit Contents

The EK-5209-5 Evaluation Kit contains the following:

- (5) iB-5209 Endpoints.
- (3) RT-5209 Routers.
- (1) GW-5209 Gateway.
- Terminal board with battery for each i-Bean Endpoint, which is factory installed on a terminal board.
- Antennas for each i-Bean Router and i-Bean Gateway.
- Battery holder and (2) AA batteries for each i-Bean Router and i-Bean Gateway (battery holder for the i-Bean Gateway is factory installed inside its enclosure).
- Regulated AC adapters for each i-Bean Router and i-Bean Gateway (AC adapter is factory installed on the i-Bean Gateway).
- (1) RS-232 serial cable for connecting the i-Bean Gateway to the host PC. This is a DB-9, male-to-female, straight though cable.
- (1) Temperature sensor assembly (for sample iB-5209 application).
- CD-ROM containing support documentation and application software, including the iB-5209 Network Monitor program and API software.

Warning

These electronic products are sensitive to electrostatic discharge (ESD). Permanent damage to these devices can result if subjected to high energy electrostatic discharges.

Proper precautions are recommended to avoid performance degradation or loss of functionality.

Host PC Requirements

The evaluation kit requires a personal computer (PC) for operation. The host PC must have the following minimal configuration:

- MS Windows XP or 2000
- RS-232 serial port
- CD-ROM drive for loading software
- Display with SVGA (800 x 600) resolution or greater

2

Kit Installation

This chapter provides the following i-Bean network installation information:

- 'Installing the i-Bean Wireless Network' on page 2-2
- 'Hardware Installation' on page 2-3
- 'iB-5209 Network Monitor Installation' on page 2-10

Installing the i-Bean Wireless Network

This section of the user's guide describes how to install the evaluation kit's hardware and software components, which are installed in the following order:

1. i-Bean Gateway (see 'i-Bean Gateway Setup (GW-5209)' on page 2-3)
2. i-Bean Routers (see 'i-Bean Router Setup (RT-5209)' on page 2-6)
3. i-Bean Endpoints (see 'i-Bean Endpoint Setup (iB-5209)' on page 2-9)
4. iB-5209 Network Monitor (see 'Installing contents of Millennial Net's iB-5209 CD-ROM' on page 2-10)
5. Applications Programming Interface (see API Functions (page A-1))

Once the hardware is set up and the software (iB-5209 Network Monitor) installed, you will launch iB-5209 Network Monitor and verify that all hardware is detected and displayed by iB-5209 Network Monitor.

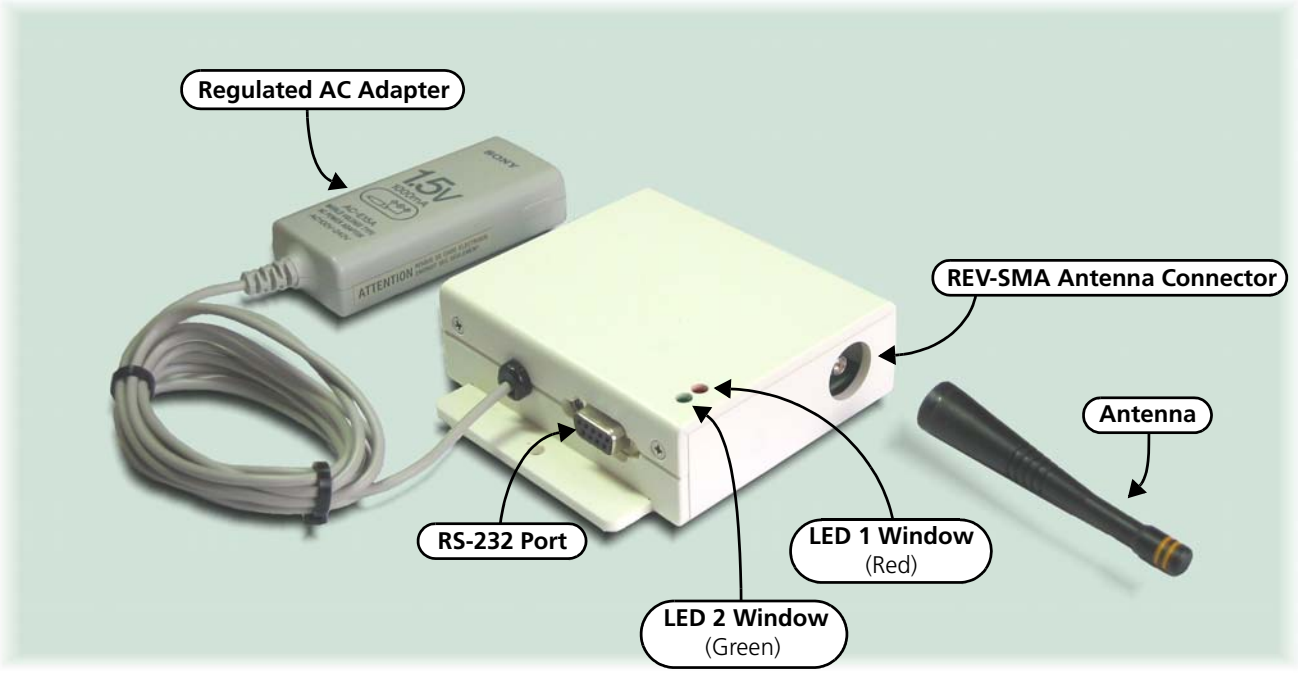
Hardware Installation

The following procedures describe in order, how to install the various hardware components of the evaluation kit. When initially setting up the hardware, it is recommended that the i-Bean Gateway, i-Bean Routers, and i-Bean Endpoints be placed close to the host PC. This will make verifying proper network installation and operation easier when first establishing a session with iB-5209 Network Monitor. The devices can then be moved away from the host PC as needed.

i-Bean Gateway Setup (GW-5209)

The i-Bean Gateway, model number GW-5209, is shipped enclosed in a case that has openings for instant access to the antenna and RS-232 connectors as shown in Figure 2-1. The case cover only needs to be removed should you decide to power the device with batteries instead of the factory-installed AC adapter, as explained in the following setup procedure.

Figure 2-1. i-Bean Gateway components (external)



During the setup procedure, refer to Figure 2-1 for location of the various i-Bean Gateway components.

To install the i-Bean Gateway:

1. Attach one of the included antennas to the REV-SMA antenna connector. The antenna screws onto the connector.

Caution

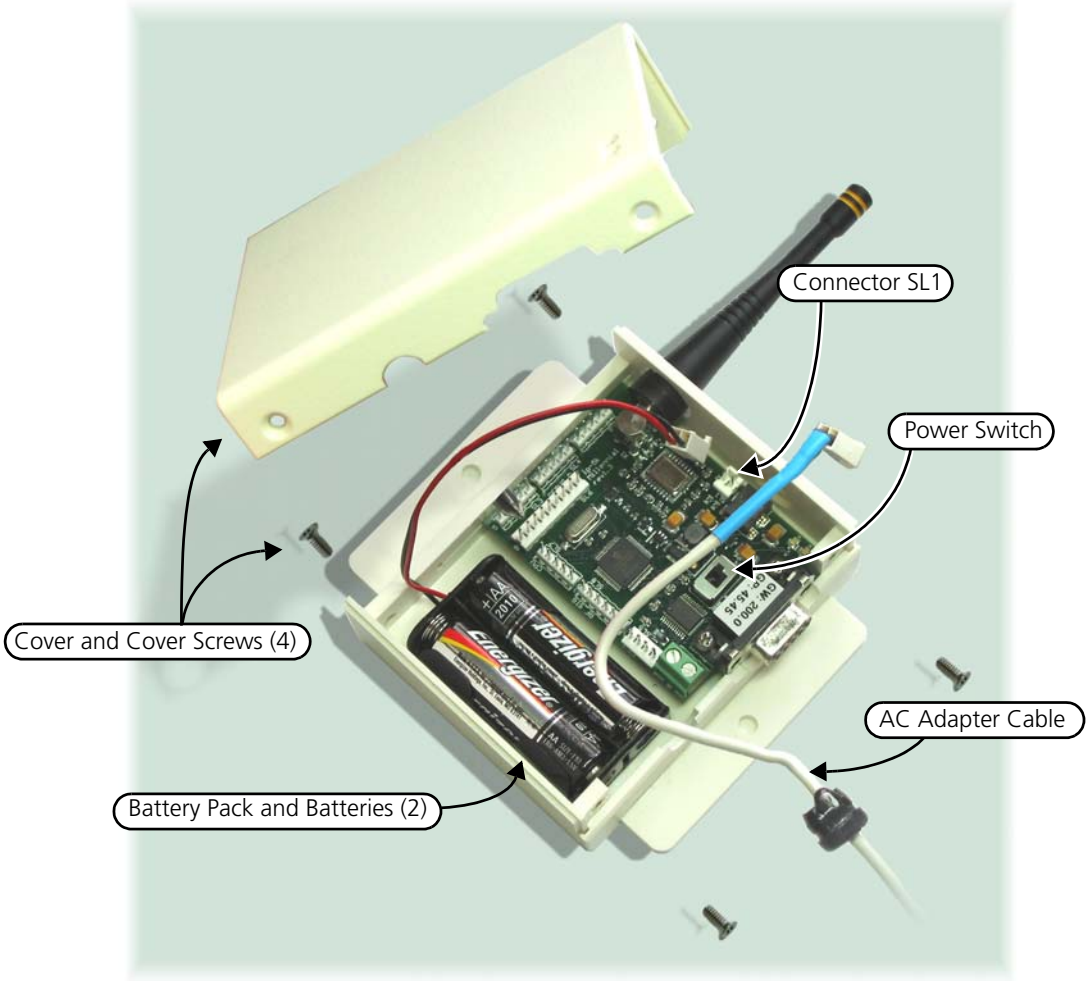
When attaching the antenna, only hand-tighten the antenna to the connector. Using excessive force may damage the connector.

2. Connect the RS-232 cable between the i-Bean Gateway and the host PC.

3. (*recommended power connection*) Plug the factory-installed AC adapter into a 110 VAC power source. The i-Bean Gateway's internal power switch is set to ON, so LED 2 (green) illuminates as soon as power is applied (see Table 2-1 on page 5 for LED operation).

4. (*optional power connection*) This step is only required when using the internal battery pack to supply power the device. To install batteries and connect the battery pack (see Figure 2-2):
   a. Remove the four philips head screws securing the cover to the base, then remove the cover.
   b. Remove the AC adapter connector from **SL1** and remove the AC adapter assembly.
   c. Turn the Power Switch **OFF**.
   d. Install two AA batteries into the battery pack (observe polarity).
   e. Connect the battery pack to keyed connector **SL1**.
   f. Turn the Power Switch **ON**. LED 2 illuminates.
   g. Replace the cover and secure in place with the four philips head screws.

Figure 2-2.Powering the i-Bean Gateway with batteries



i-Bean Gateway status LED operation

Table 2-1 describes how the status LEDs on the i-Bean Gateway behave.

Table 2-1. i-Bean Gateway Status LEDs

Device Status	LED1 (red)	LED2 (green)
• Power off	Off	Off
• Power on • Host PC connected • iB-5209 Network Monitor *not* running	N/A	Illuminates; flashing (0.5 second On/ 0.5 second Off)
• Power on • Host PC connected • iB-5209 Network Monitor running	N/A	Illuminates; solid
• Power on • Network traffic exists*	Illuminates; flashing (indicating network activity)	Illuminates; solid

* A device's LED1 will flash when detecting valid packets (packets destined for device) and may also flash when detecting invalid packets (packets destined for other devices) or environmental noise. Only valid packets are processed by the device.

i-Bean Router Setup (RT-5209)

The i-Bean Routers, model number RT-5209, are shipped without cases to provide full access to the various I/O connectors. (While the i-Bean Gateway has similar connectors, they are only functional on the i-Bean Router.) Refer to Figure 2-3 on page 2-7 for locations of the various i-Bean Router switches and connectors described in the following procedure.

Caution

A jumper is factory-installed on JP1 that connects pins 2 and 3 together. Do not remove or change the position of this jumper as it will cause the device to malfunction.

To install an i-Bean Router:

1. Attach one of the included antennas to the REV-SMA antenna connector. The antenna screws onto the connector.

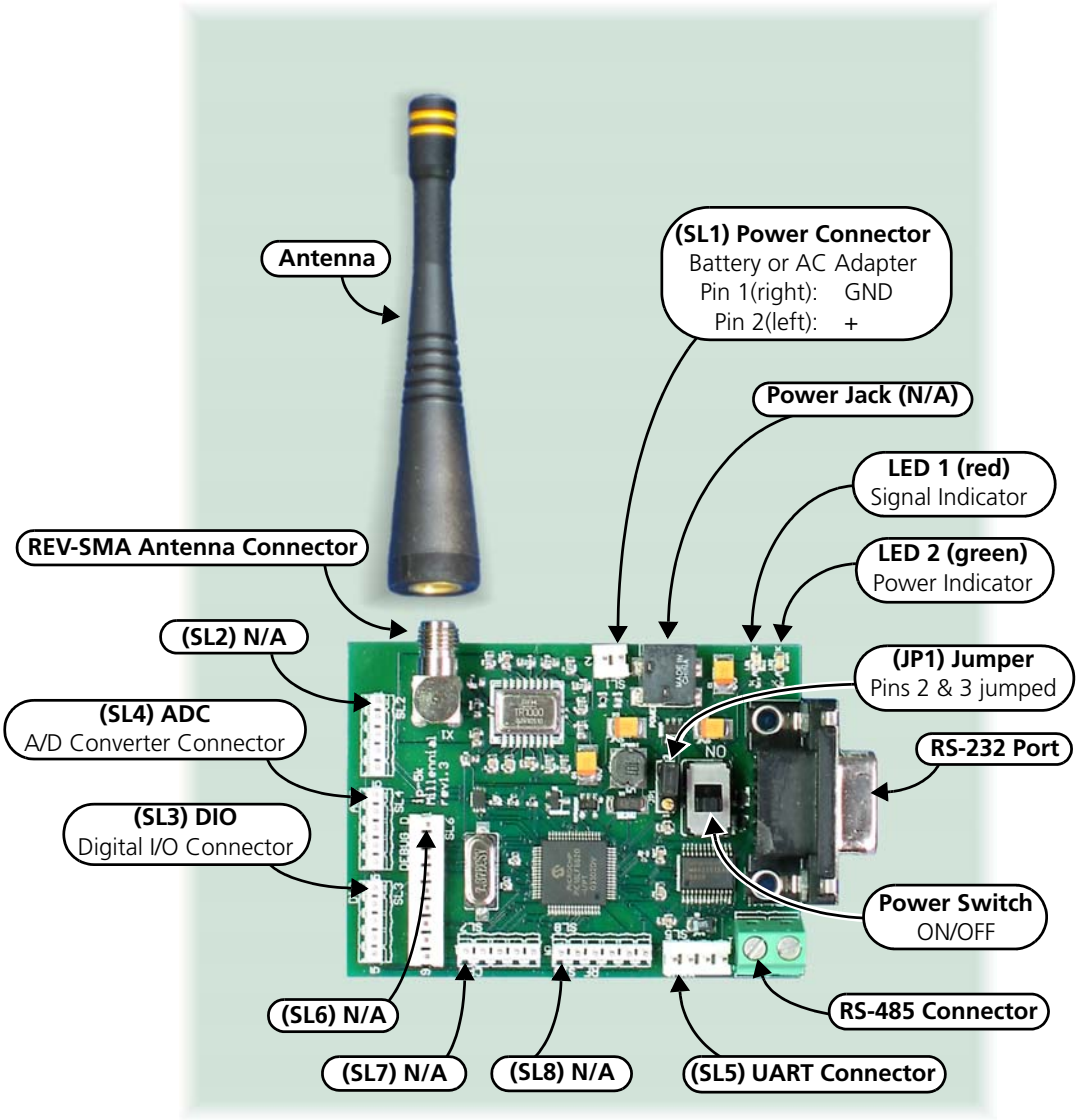
Caution

When attaching the antenna, only hand-tighten the antenna to the connector. Using excessive force may damage the connector.

2. Turn the Power Switch **OFF**.
3. Connect one of the following power sources supplied with the kit:
   - **Regulated AC Adapter** (recommended): Plug the regulated AC adapter into keyed connector **SL1** and a 110 VAC power source.
   - **Batteries**: To install batteries and connect the battery pack:
     a. Install two AA batteries into the battery pack (observe polarity).
     b. Connect the battery pack to keyed connector **SL1**.
4. Turn the Power Switch **ON**. LED 2 (green) illuminates, indicating power is applied to the device (see Table 2-2 on page 8 for LED operation).
5. Repeat Steps 1–4 for each i-Bean Router in the kit.

Note: Configuring and connecting an i-Bean Endpoint or i-Bean Router to external devices via their digital or analog I/O connectors is discussed in Chapter 3, "iB-5209 Network Monitor Operations".

Figure 2-3. i-Bean Router components



Notes: 1. Callouts for **SL1–8** and **RS-485** point to pin 1 of each connector
2. Connectors labeled as **N/A** are for Millennial Net use only.

i-Bean Router status LED operation

Table 2-1 describes how the status LEDs on the i-Bean Router behave.

Table 2-2. i-Bean Router Status LEDs

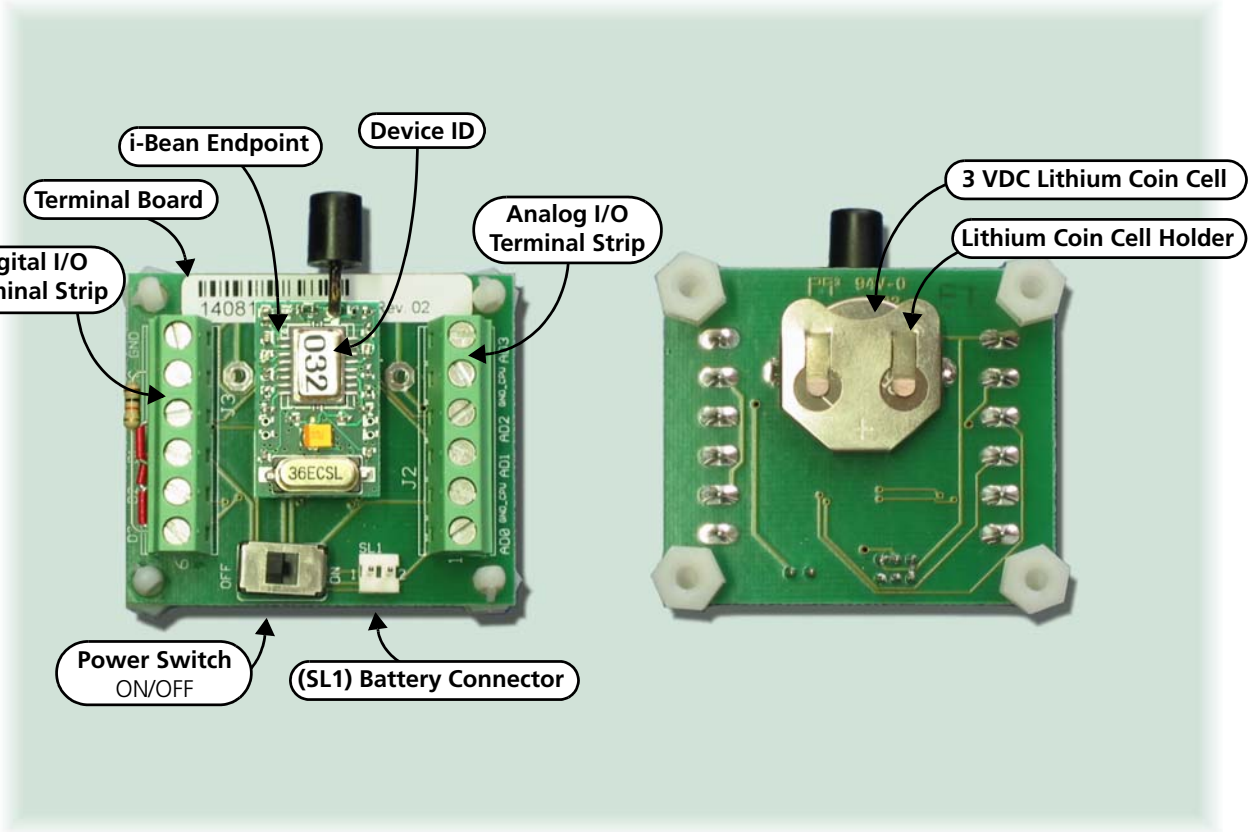
Device Status	LED1 (red)	LED2 (green)
• Power off	Off	Off
• Power on • i-Bean Gateway cannot be found (off network)	N/A	Illuminates; flashing (1 second On/ 1 second Off)
• Power on • i-Bean Gateway found (on network)	N/A	Illuminates; solid
• Power on • Network traffic exists*	Illuminates; flashing (indicating network activity)	Illuminates; solid

* A device's LED1 will flash when detecting valid packets (packets destined for device) and may also flash when detecting invalid packets (packets destined for other devices) or environmental noise. Only valid packets are processed by the device.

i-Bean Endpoint Setup (iB-5209)

The i-Bean Endpoints, model number iB-5209, are mounted to terminal boards as shown in Figure 2-4. A terminal board provides easy access to I/O connections and the power switch. The terminal board also contains a battery holder and battery for supplying power to the i-Bean Endpoint. The label on top of the i-Bean Endpoint denotes the last three digits of the device ID assigned to the i-Bean Endpoint.

Figure 2-4. i-Bean Endpoint and terminal board (top and bottom views)



To provide power to the i-Bean Endpoint, a 3 VDC lithium coin cell (CR2032 type) is provided and already installed in the battery holder on the back of the terminal board.

To activate an i-Beam Endpoint:

- Turn the terminal board's power switch **ON** (refer to Figure 2-4). Repeat this step for each i-Bean Endpoint.

Note: Configuring and connecting an i-Bean Endpoint or i-Bean Router to external devices via their digital or analog I/O connectors is discussed in Chapter 3, "iB-5209 Network Monitor Operations".

iB-5209 Network Monitor Installation

The procedures in this section describe how to do the following:

1. Use the CD-ROM shipped with the evaluation kit to install the following Millennial Net items on the host PC:
   - iB-5209 Network Monitor
   - API examples
   - API documentation
   - iB-5209 i-Bean Endpoint Data Sheet
   - GW-5209 i-Bean Gateway and RT-5209 i-Bean Router Data Sheet
   - EK-5209-5 Evaluation Kit User's Guide
2. Open an iB-5209 Network Monitor session.

The software installation procedure utilizes an InstallShield Wizard that will guide you through the installation process. When the process is complete, an iB-5209 Network Monitor shortcut icon is also added to the host PC's desktop.

Note: Before installing and launching iB-5209 Network Monitor, be sure that all i-Bean network hardware components are installed and turned on as described in 'Hardware Installation' on page 2-3.

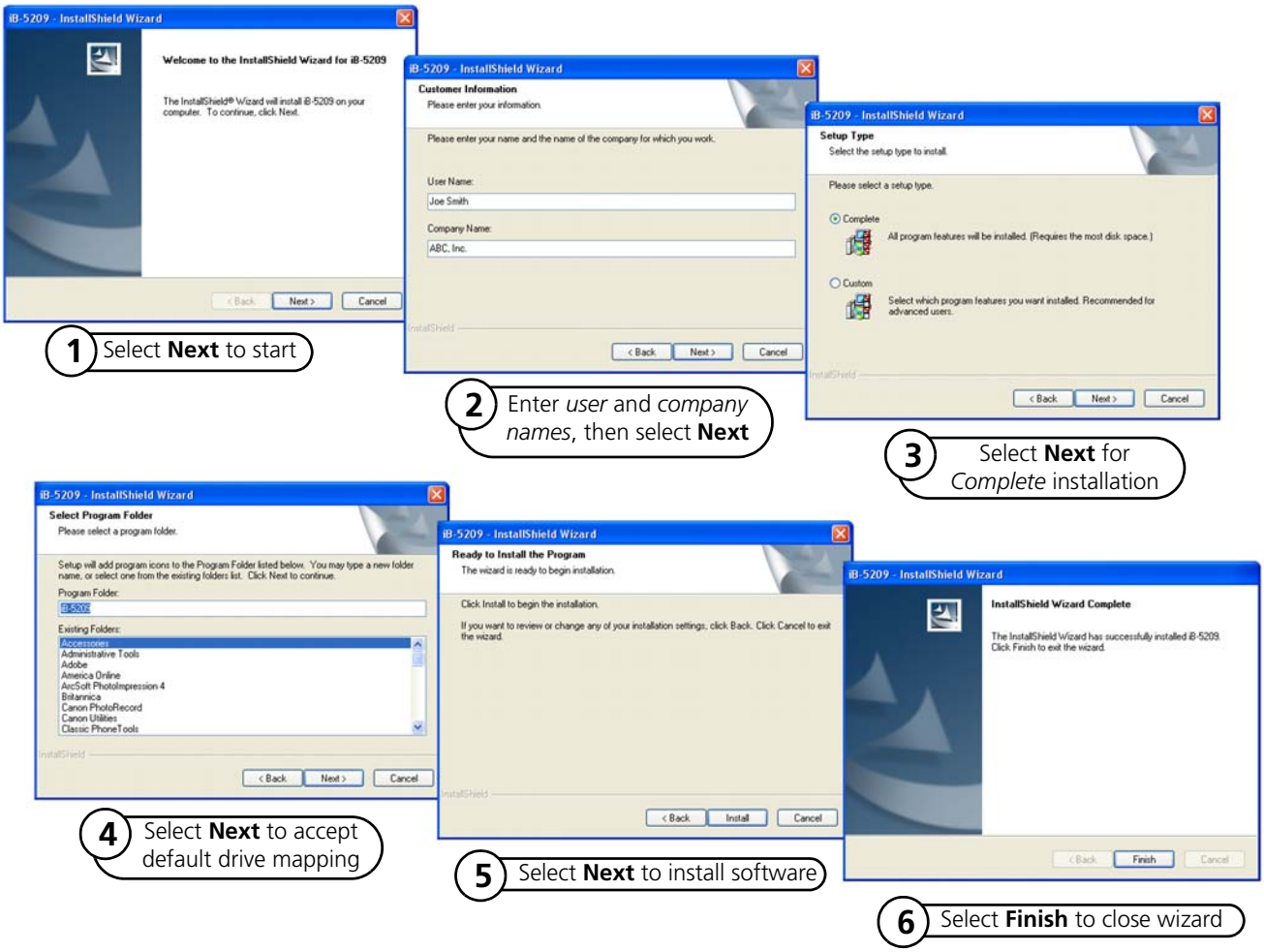
Installing contents of Millennial Net's iB-5209 CD-ROM

To install the software contained on the CD-ROM (see Figure 2-5 on page 2-11):

1. Insert the *iB-5209 Evaluation Kit CD* into the host PC's CD-ROM drive. The Autorun feature launches the InstallShield Wizard, displaying the wizard's first screen. Select **Next** to start the installation procedure.
2. Enter a *User Name* and *Company Name*, then select **Next**.
3. To install the complete installation package (default setting), select **Next**.
4. To have iB-5209 Network Monitor installed in the Programs folder on the C: drive (default setting), select **Next**.
5. The InstallShield Wizard displays a screen that allows you to change any of the selected installation settings before the actual software installation process begins. Select **Next** to continue. The software installation process starts.
6. Select **Finish** to complete the process and close the InstallWizard Shield.

Proceed to 'Launching iB-5209 Network Monitor' on page 2-12.

Figure 2-5. Installing Millennial Net's CD-ROM contents



Launching iB-5209 Network Monitor

To launch iB-5209 Network Monitor and verify proper communication with the i-Bean network nodes (see Figure 2-6):

1. Do one of the following to launch iB-5209 Network Monitor:

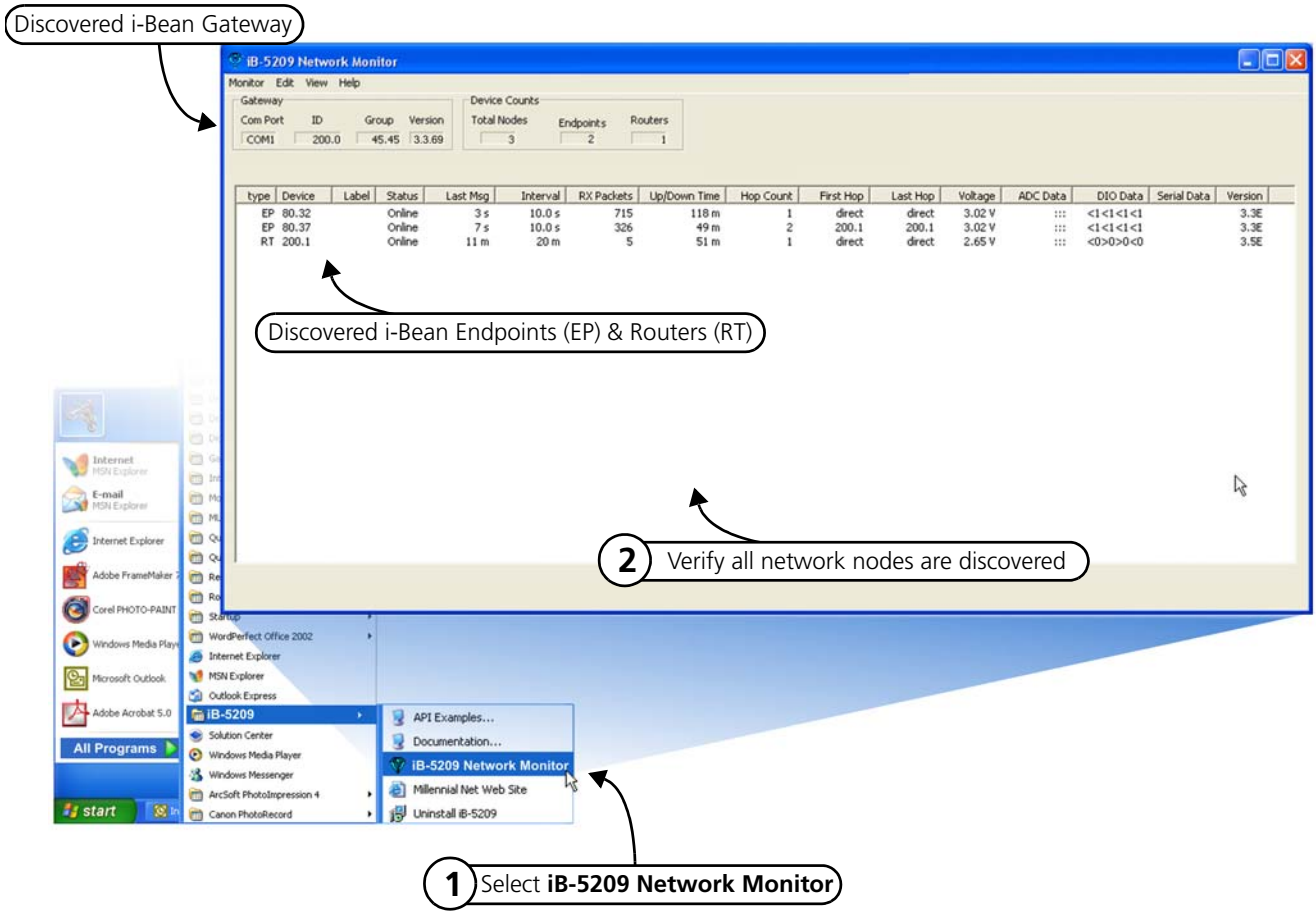
   

   – Double-click on the desktop's iB-5209 Network Monitor icon.

   – From the Windows' taskbar, select:
   Start>All Programs>iB-5209>iB-5209 Network Monitor.

2. Verify that all network nodes are discovered and displayed by iB-5209 Network Monitor.

Figure 2-6. Launching iB-5209 Network Monitor



Once proper operation of the i-Bean network has been verified, proceed to Chapter 3, "iB-5209 Network Monitor Operations" for an overview of the GUI and details on how to use it to configure the operation of your i-Bean network.

3

iB-5209 Network Monitor Operations

This chapter provides the following iB-5209 Network Monitor information:

- 'iB-5209 Network Monitor Overview' on page 3-2
- 'Setting Thread Priority' on page 3-6
- 'Configuring a Node's Operation' on page 3-7
- 'Creating an Event Log File' on page 3-21
- 'Configure Persistence Attributes' on page 3-22
- 'Configure Serial and ADC Data Formats' on page 3-23
- 'Select Com Port on Host PC' on page 3-24
- 'View Monitor Statistics' on page 3-25
- 'View Contents of Event Log File' on page 3-26
- 'Enable Multiple Capture' on page 3-28

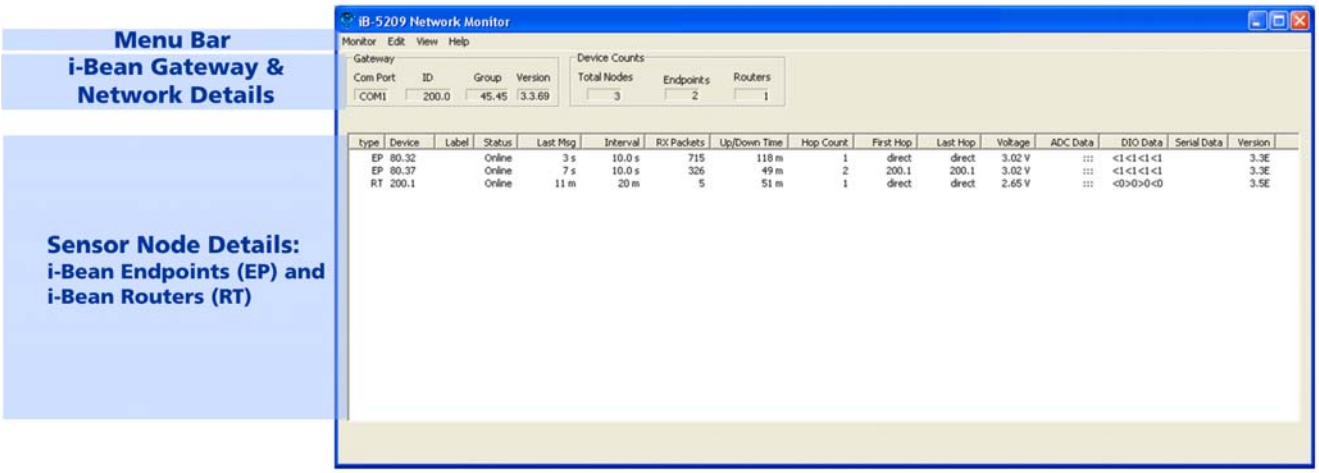
iB-5209 Network Monitor Overview

Millennial Net's iB-5209 Network Monitor is a monitoring and management system for iB-5209 networks. This management tool will discover and display active i-Bean Routers in the vicinity of the i-Bean Gateway, and i-Bean Endpoints in the range of the i-Bean Gateway and i-Bean Routers as shown in Figure 3-1. iB-5209 Network Monitor displays the *Group ID* and *Device ID* of the i-Bean Gateway and will display only i-Bean Endpoints and i-Bean Routers that have the same Group ID as the i-Bean Gateway. (For information on opening a iB-5209 Network Monitor session, see 'Launching iB-5209 Network Monitor' on page 2-12.)

Using iB-5209 Network Monitor, a number of the monitoring features may be observed:

- **Any of the i-Bean Endpoints can be moved, and as long as they are within the range of an i-Bean Router or the i-Bean Gateway, connectivity will be maintained seamlessly.** Any of the i-Bean Routers and even the i-Bean Gateway can be moved while operating, and all routes will automatically adapt to their new locations.
- **The i-Bean network routing protocol always seeks to route data using the most reliable RF links and using the fewest hops.** The network protocol will change the route when an RF link in the route is deemed unreliable. This can be seen in the iB-5209 Network Monitor. For example, the i-Bean Router IDs used for the first and last hops may change from time to time even when the i-Bean Endpoint is stationary, due to environmental interference.
- **If any of the i-Bean Routers runs out of battery power or is turned off, all routes that went through that i-Bean Router will be reconfigured—all i-Bean Endpoints communicating with that i-Bean Router will still be connected to other i-Bean Routers without any disruption or loss of packets.** However, if an i-Bean Endpoint exceeds the range of the network due to the loss of an i-Bean Router, then the i-Bean Endpoint will be displayed as Offline or removed from the display (depending on how the Persistence function is configured).

Figure 3-1. Sample iB-5209 Network Monitor window



As shown in Figure 3-1, the main window is divided into the following sections:

Menu Bar

From the menu bar, system users access the following:

- **Monitor**

  This menu option provides access to the following functions:

  – **Thread Priority**: This setting refers to the priority level of the iB-5209 Network Monitor program in a MS Windows operating system environment. For details, see 'Setting Thread Priority' on page 3-6.

  – **Exit**: Ends the session and closes iB-5209 Network Monitor.

- **Edit**

  – **Devices**: Displays the Edit Device window used to configure a node's sampling interval time, I/O interfaces, and start/stop recording I/O information to a log file. For details, see 'Configuring a Node's Operation' on page 3-7.

  – **Labels**: Assign user-defined names to i-Bean Endpoints and i-Bean Routers on the network. For details, see 'Labeling i-Bean Endpoint or i-Bean Router' on page 3-20.

  – **Logging**: Create a log file of reported network events, such as reported up/down events and changes to voltages or routes. For details, see 'Creating an Event Log File' on page 3-21.

  – **Persistence**: Stop monitoring/displaying offline i-Bean Endpoints and i-Bean Routers. For details, see 'Configure Persistence Attributes' on page 3-22.

  – **Data Display**: Configure the following I/O data formats:

    - *Serial Data Format*: Define format of displayed serial data (ASCII/Hex/Decimal)
    - *ADC Data Format*: Define format of displayed ADC data (Voltage/Raw Data)

    For details, see 'Configure Serial and ADC Data Formats' on page 3-23.

  – **Com Port**: Select serial port on Host PC to use for i-Bean Gateway connection. For details, see 'Select Com Port on Host PC' on page 3-24.

  – **All Sampling Intervals**: Configure all network nodes with the same sampling interval time. For details, see 'Configuring Sample Interval of all Network Nodes' on page 3-11.

- **View**

  – **Monitor Statistics**: Open Monitor Statistics window, displaying RX/TX packet and byte information.

  – **Log**: Display content of log file.

- **Help**

  – **About**: Displays iB-5209 Network Monitor revision level information.

Gateway

This section displays the following information on the i-Bean Gateway connected to the host PC's RS-232 port:

- **Com Port**: Host PC's RS-232 port connected to i-Bean Gateway.
- **ID**: Device identifier assigned to the i-Bean Gateway. The ID consists of two octets (A.B), where each octet's value = 000 to 255.
- **Group**: Group identifier assigned to the i-Bean Gateway. All i-Bean Routers and i-Bean Endpoints with the same Group identifier will communicate with the displayed i-Bean Gateway.
- **Version**: Version of firmware loaded on the i-Bean Gateway.

Device Counts

This section displays the following information on the discovered network nodes:

- **Total Nodes**: Total combined number of discovered i-Bean Endpoint and i-Bean Router nodes.
- **Endpoints**: Total number of i-Bean Endpoint nodes.
- **Routers**: Total number of discovered i-Bean Router nodes.

Node Details

This section displays the following information related to the i-Bean Endpoint and i-Bean Router nodes on the network:

- **Type**: This column lists all nodes discovered on the network that are assigned the same Group ID as the displayed i-Bean Gateway. Nodes displayed here include i-Bean Endpoints (EP) and i-Bean Routers (RT).
- **Device**: Unique identifier assigned each node. The identifier consists of two octets (A.B), where each octet contains a value between 000 and 255.
- **Label**: User-defined name assigned to node.
- **Status**: Current status of the device:
  - **Online**: The node is communicating with the i-Bean Gateway.
  - **Offline**: The i-Bean Gateway can no longer communicate with the node.
  - **Queued**: The node, when it first comes into the network, is waiting to be acknowledged by the i-Bean Gateway.
  - **Late**: Packet from node is delayed (possible route interference).
  - **Refresh**: The node is being updated with a new operating state.
- **Last Msg**: Time elapsed since last packet was received from the node. Time is displayed in seconds (s).

- **Interval**: Time of the last message generated by a node. This value is synchronized with the sampling interval of the node. Time is displayed in either seconds (s) or minutes (m).
- **RX Packets**: Number of packets successfully delivered to iB-5209 Network Monitor from a node since the node was detected by the i-Bean Gateway. The counter is reset if one of the following actions occur:
  - The node is powered down.
  - The i-Bean Gateway is powered down.
  - An i-Bean Router used as a network hop is powered down.
  - iB-5209 Network Monitor program is restarted.
- **Up/Down Time**: Time since the node was first detected by the i-Bean Gateway.
- **Hop Count**: Number of network node hops taken by a packet delivered from a node to the i-Bean Gateway. For example: i-Bean Endpoint—i-Bean Gateway = 1 hop, i-Bean Endpoint—i-Bean Router—i-Bean Gateway = 2 hops (each additional i-Bean Router will add another hop).
- **First Hop**: Device ID of the first i-Bean Router on the path used by a packet to get to the i-Bean Gateway. If no i-Bean Router was used, then *Direct* will display, indicating the device is communicating directly with the i-Bean Gateway.
- **Last Hop**: Device ID of the last i-Bean Router on the path used by a packet to get to the i-Bean Gateway. If no i-Bean Router was used, then *Direct* will display, indicating the device is communicating directly with the i-Bean Gateway.
- **Voltage**: DC voltage level of the node's power source in volts.
- **ADC Data**: Input voltages on pins used for analog-to-digital conversion operation.
- **DIO Data**: Digital information on pins used for digital I/O operation.
- **Serial Data**: Input serial data information when configured for serial operation.
- **Version**: Version of firmware loaded on the node.

Setting Thread Priority

When iB-5209 Network Monitor is installed on a host PC using a MS Windows operation system, the priority level the program can be adjusted. The setting of Thread Priority affects how the iB-5209 Network Monitor program is handled by the Windows operating system, which in turn affects the reliability of data communications among the i-Bean network nodes as well as between the i-Bean Gateway and the host PC. When Thread Priority is set to **High**, iB-5209 Network Monitor will be given high priority, and therefore data communications will be most reliable; however, the performance of certain PC peripherals such as the mouse may be affected. Therefore, it is recommended that the Thread Priority setting be set to **Normal** for most applications. A setting of **High** should be used for tasks where communication reliability is critical.

To set the Thread Priority level:

1. Select **Monitor**>**Thread Priority** from the menu bar. A list of priority levels displays.
2. Select the desired priority level:
   - **Highest**
   - **Normal** (default setting; recommended)
   - **Lowest**
   - **Idle**

Configuring a Node's Operation

From iB-5209 Network Monitor's menu bar, selecting **Edit>Devices** displays the Edit Device window used to configure the operation of an i-Bean Endpoint or i-Bean Router. Figure 3-2 shows a sample of the Edit Device window.

Figure 3-2. iB-5209 Network Monitor's Edit Device window

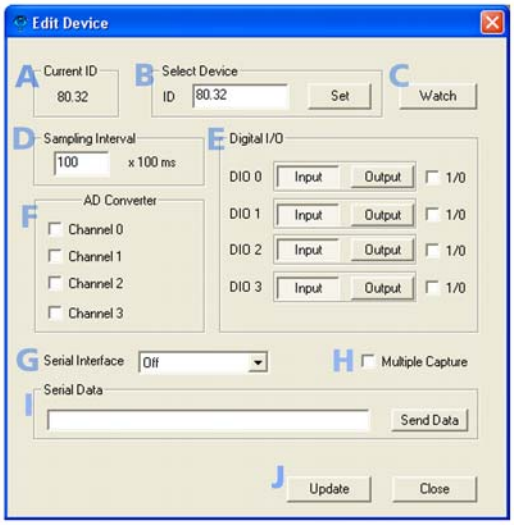


Table 3-1 describes the functions of the various sections of the window as shown in Figure 3-2.

Table 3-1. Edit Device window functions

Item	Description	Function
A	Current ID	This is the Device ID of the node currently selected for configuring.
B	Select Device	Use this panel to select another node to configure by entering the device ID of the desired node, then selecting **Set**.
C	Watch	This option opens a new widow that displays information relating to the the node's various interfaces, including analog and digital I/O configuration states and packets received/sent. For details, see 'Using Watch function to display current I/O information' on page 3-9.
D	Sampling Interval	This functions configures how often the node transmits a 'heart beat' data packet. For details, see 'Configuring Sample Interval of Single Node' on page 3-11.
E	Digital I/O	This panel is used to control the states of the I/O pins associated with digital I/O channels D0–D3. For details, see 'Configuring Digital I/O Operation' on page 3-12

Table 3-1. Edit Device window functions (continued)

Item	Description	Function
F	AD Converter	This panel is used to control the states of the AD (Analog-to-Digital) Converter channels. For details, see 'Configuring AD (analog-to-digital) Converter Operation' on page 3-17.
G	Serial Interfaces	This panel is used to select a serial I/O operation for a device: Digital UART, RS-232 (RT-5209 only), or RS-485 (RT-5209 only). Selecting serial operation disables digital I/O functionality. For details, see the following: • 'Configuring UART Operation' on page 3-15 • 'Configuring RS-232 Operation (RT-5209 only)' on page 3-18 • 'Configuring RS-485 Operation (RT-5209 only)' on page 3-19
H	Multiple Capture	Multiple capture is a special feature that enables iB-5209 Network Monitor to report the surrounding router information of a given device (an i-Bean Endpoint or an i-Bean Router that has endpoint functionality). For details, see 'Enable Multiple Capture' on page 3-28.
I	Serial Data	This panel, which is only used if the node is configured for serial operation, is used to send serial data to the node and configure the length of the string sent. For details, see the following: • 'Configuring UART Operation' on page 3-15 • 'Configuring RS-232 Operation (RT-5209 only)' on page 3-18 • 'Configuring RS-485 Operation (RT-5209 only)' on page 3-19
J	Update/Close	• **Update**: Updates the selected device with any changes made to its configuration. • **Close**: Closes the Edit Device window. If changes where made to the device configuration and Update was not selected before selecting Close, the changes will be ignored.

Using Watch function to display current I/O information

To display the current status information relating to the node's interfaces (see Figure 3-3):

1. Select **Edit**>**Devices**. The Edit Device window opens.
2. Using the *Select Device* panel, enter the device ID of the desired node, then select **Set**. The node's device ID displays in the *Current ID* panel.
3. Select **Watch**. The **Watch** window opens, displaying the selected node's I/O status information.

Figure 3-3. Displaying I/O information using Watch function

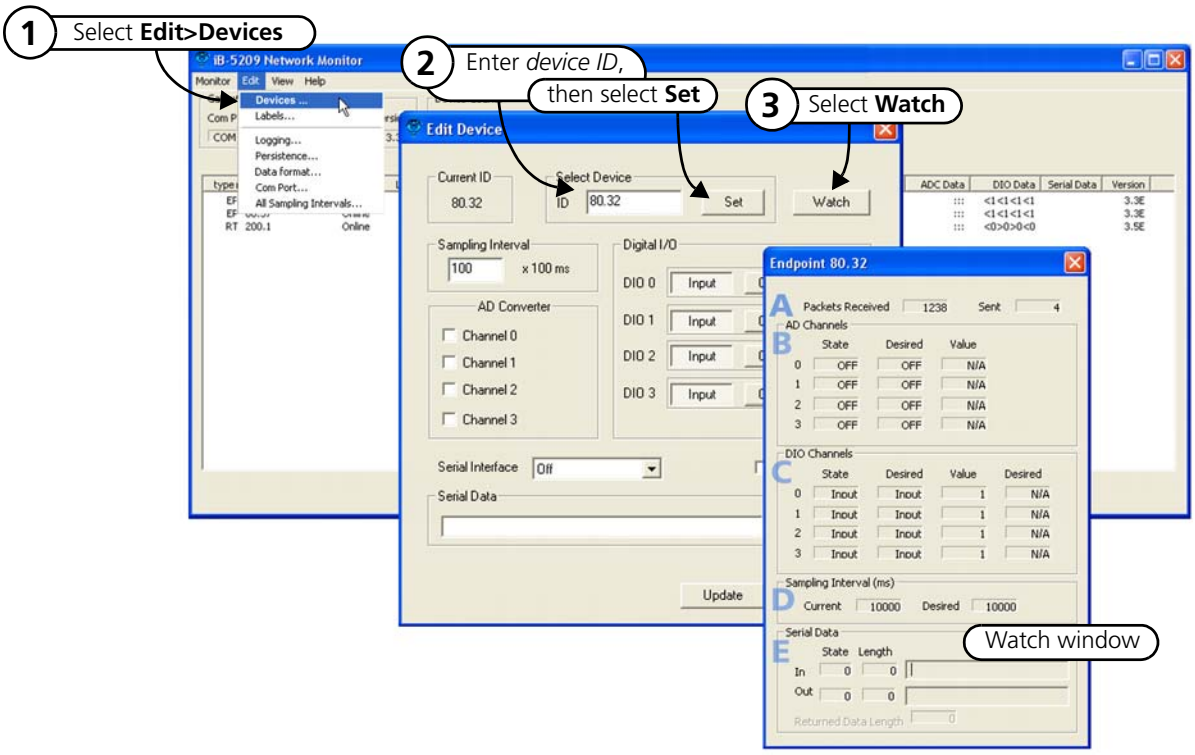


Table 3-2 describes the functions of the various sections of the Watch window as shown in Figure 3-3.

Table 3-2. Watch window functions

Item	Description	Function
A	Packets Received/Sent	Total packets received from /transmitted to the node.
B	AD Channels	This panel displays the following information for each analog-to-digital channel (0–3): • **State**: Current on/off state of the channel • **Desired**: Desired on/off state of channel, which can be changed using the **Edit**>**Devices** option. • **Value**: Numeric values of the channel.

Table 3-2. Watch window functions (continued)

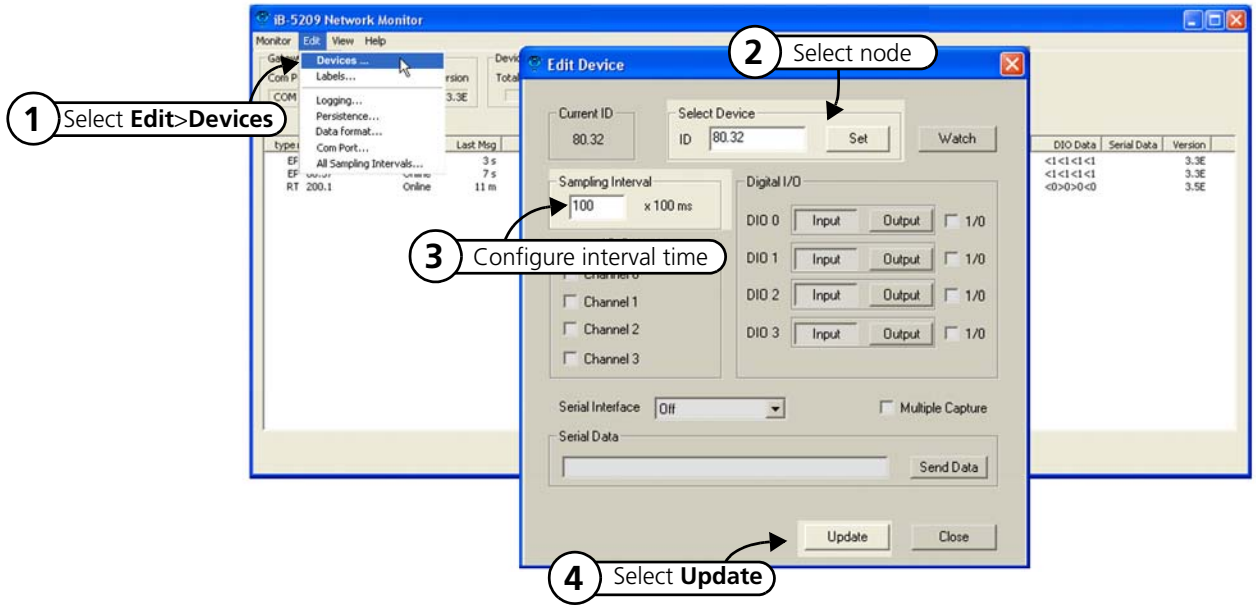
Item	Description	Function
C	DIO Channels	This panel displays the following information for each digital channel (0–3): • **State**: Current input/output state of the channel • **Desired**: Desired input/output state of channel, which can be changed using the **Edit**>**Devices** option. • **Value**: I/O values of the channel (1 or 0). • **Desired**: Desired I/O value of channel.
D	Sampling Interval	This panel displays the following sampling interval information: • **Current**: Currently configured setting. • **Desired**: Desired setting, which can be changed using the **Edit**>**i-Bean** option (see 'Configuring Sample Interval of Single Node' on page 3-11).
E	Serial Data	This panel displays the following serial data information of data received from the node (In) or transmitted to the node (Out): • **State**: State of the serial data function (1 or 0) • **Length**: Length of string received/transmitted.

Configuring Sample Interval of Single Node

To configure the time interval between data packets transmitted by a node (see Figure 3-4):

1. Select **Edit**>**Devices**. The Edit Device window displays.
2. From the *Select Device* panel, enter the device ID of the desired node and click **Set**. The Edit Device window is now ready to make any changes to the selected node.
3. Using the *Sampling Interval* panel, enter the interval as a multiple of 100 milliseconds. For example, for an interval of 10 seconds, enter 100.
4. Select **Update**. The node's configuration is updated.

Figure 3-4. Configuring sample interval of single node

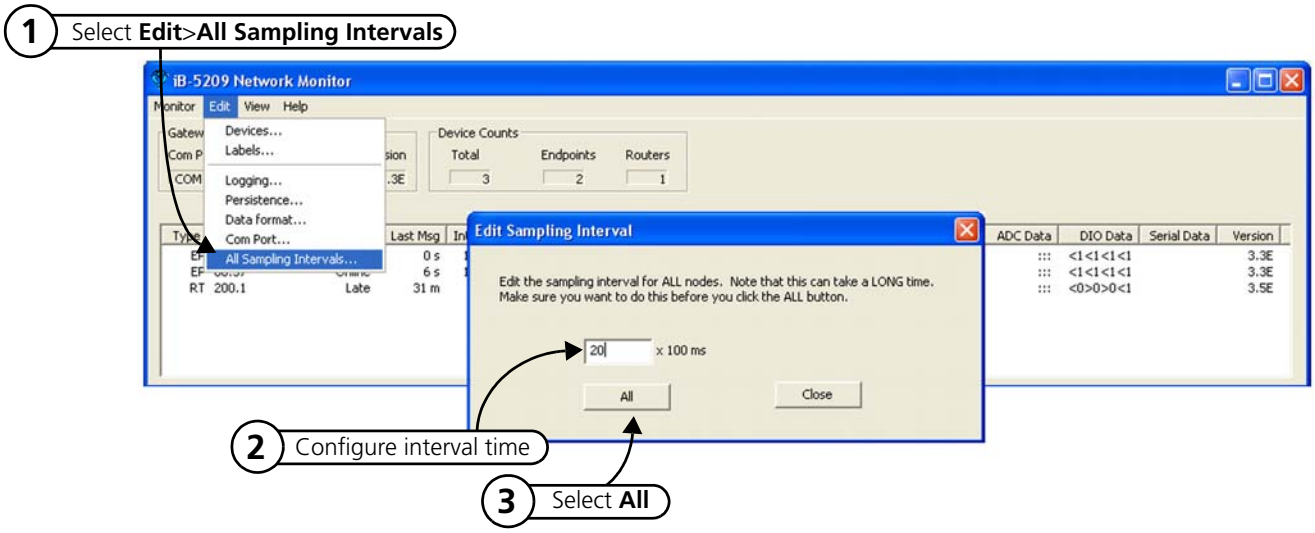


Configuring Sample Interval of all Network Nodes

To configure all nodes on the network with the same sampling interval (see Figure 3-5):

1. Select **Edit**>**All Sampling Intervals**. The Edit Sampling Interval window displays.
2. Enter the interval as a multiple of 100 milliseconds. For example, for an interval of 10 seconds, enter 100.
3. Select **All**. All node's on the network are configured with the same sampling interval rate.

Figure 3-5. Configuring sample interval of all nodes



Configuring Digital I/O Operation

The following procedure describes the steps that need to be taken to set up the hardware and configure an i-Bean Endpoint or i-Bean Router for digital I/O operation (see Figure 3-6).

Note: (*i-Bean Endpoints only*) There are factory-installed jumpers on the digital I/O terminal connectors (D0–D3, VCC). When the digital I/O channels are configured as inputs, these pins must be connected to a logic high or logic low level for optimal battery life. If left open (in an indeterminate state), the i-Bean Endpoint will consume excessive power.

Remove the factory-installed jumpers when connecting an external device to the digital I/O terminal connectors.

Digital Input Setup

1. Connect the digital source signals to the connectors (D0–D3) and Ground (GND) of:
   - i-Bean Endpoint's digital terminal block (located on terminal board).
   - i-Bean Router's terminal pins (connector SL3).

   See Figure 3-6 on page 3-14 for connector locations.
2. From iB-5209 Network Monitor, select **Edit**>**Devices**. The Edit Device window displays.
3. Enter the device ID of the desired node and click **Set**. The Edit Device window is now ready to make any changes to the selected node.
4. From the *Digital I/O* panel, select **Input** on each of the desired digital channels to use as inputs (DIO 0—DIO 3), then select **Update**.

   iB-5209 Network Monitor displays the digital information for the node in the Digital I/O Data column, where **n>** = output channel and **n<** = input channel (n = 1 or 0).

Digital Output Setup

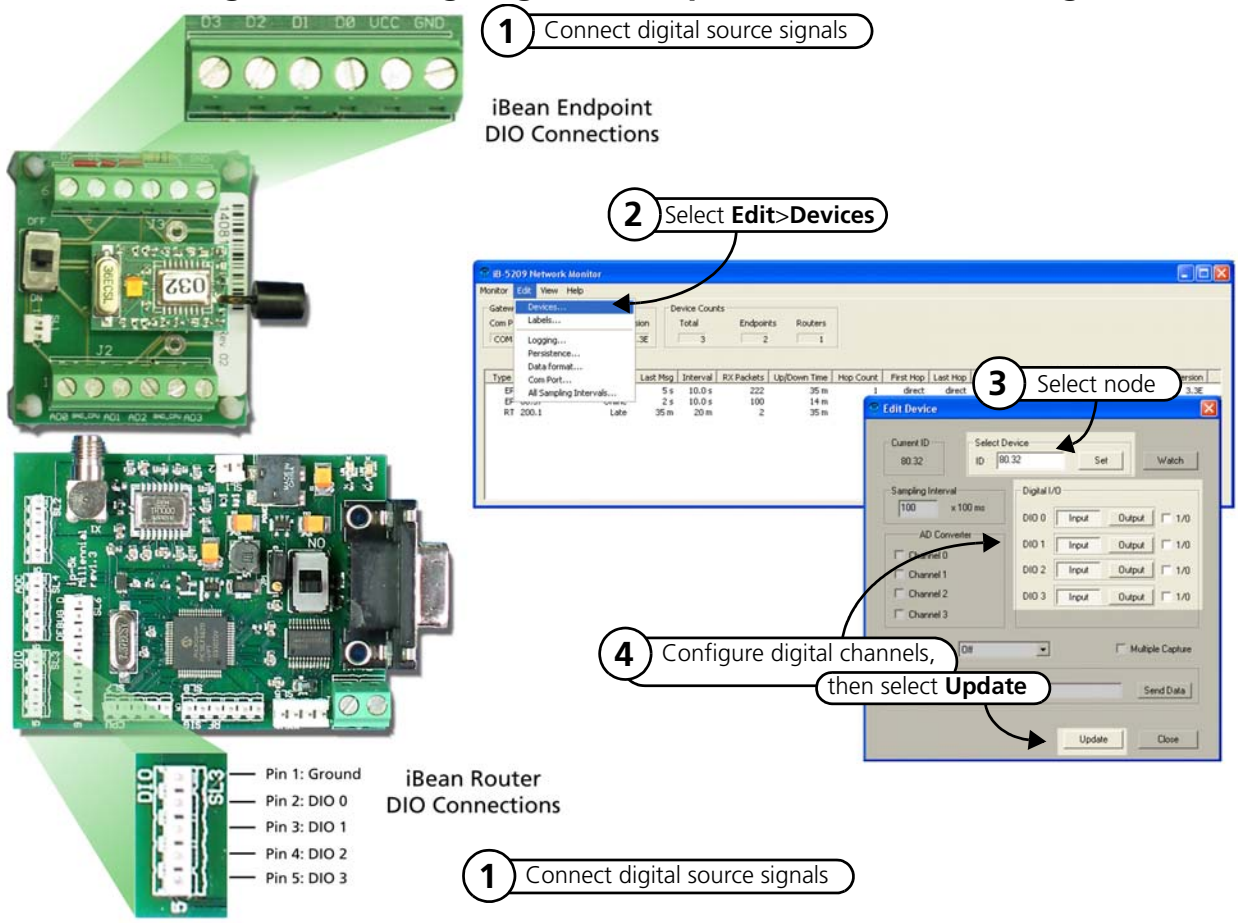
1. Connect the digital signal destination devices to the connectors (D0–D3) and ground (GND) of the following:
   - i-Bean Endpoint's digital terminal block (located on terminal board).
   - i-Bean Router's terminal pins (connector SL3).

   See Figure 3-6 on page 3-14 for connector locations.
2. From iB-5209 Network Monitor, select **Edit**>**Devices**. The Edit Device window displays.
3. Enter the device ID of the desired node and click **Set**. The Edit Device window is now ready to make any changes to the selected i-Bean Endpoint.
4. From the *Digital I/O* panel, select **Output** on each of the desired digital channels to use as outputs (DIO 0—DIO 3).
5. Set the output signal high or low using the **1/0** box located next to the desired **Output** button:
   - Selected = 1 (output is set high)
   - Not selected = 0 (output is set low)
6. Select **Update**.

   iB-5209 Network Monitor displays the digital information for the node in the Digital I/O Data column, where **n>** = output channel and **n<** = input channel (n = 1 or 0).

Note: Input signals should not be applied when the i-Bean node is switched off. Since the node is an extremely low power device, it's possible that the input signal voltages will keep the microcontroller active, preventing it from resetting properly when switched back on. Also, when switched off, the terminal board will ground certain pins, which can cause excessive current drain for the external peripheral connected to it.

Figure 3-6. Configuring i-Bean Endpoint/terminal board for digital I/O



Configuring UART Operation

The following procedures describes how to use the i-Bean Endpoint's digital I/O connections or the i-Bean Router's UART connector for serial/UART communications (see Figure 3-7). Refer to the technical specification sheets for the i-Bean Endpoint and i-Bean Router for additional information required when using the device for serial communications.

Note: The i-Bean Endpoint and i-Bean Router support the following serial communication parameters:
9600 bps, 8 data bits, no parity, 2 stop bits

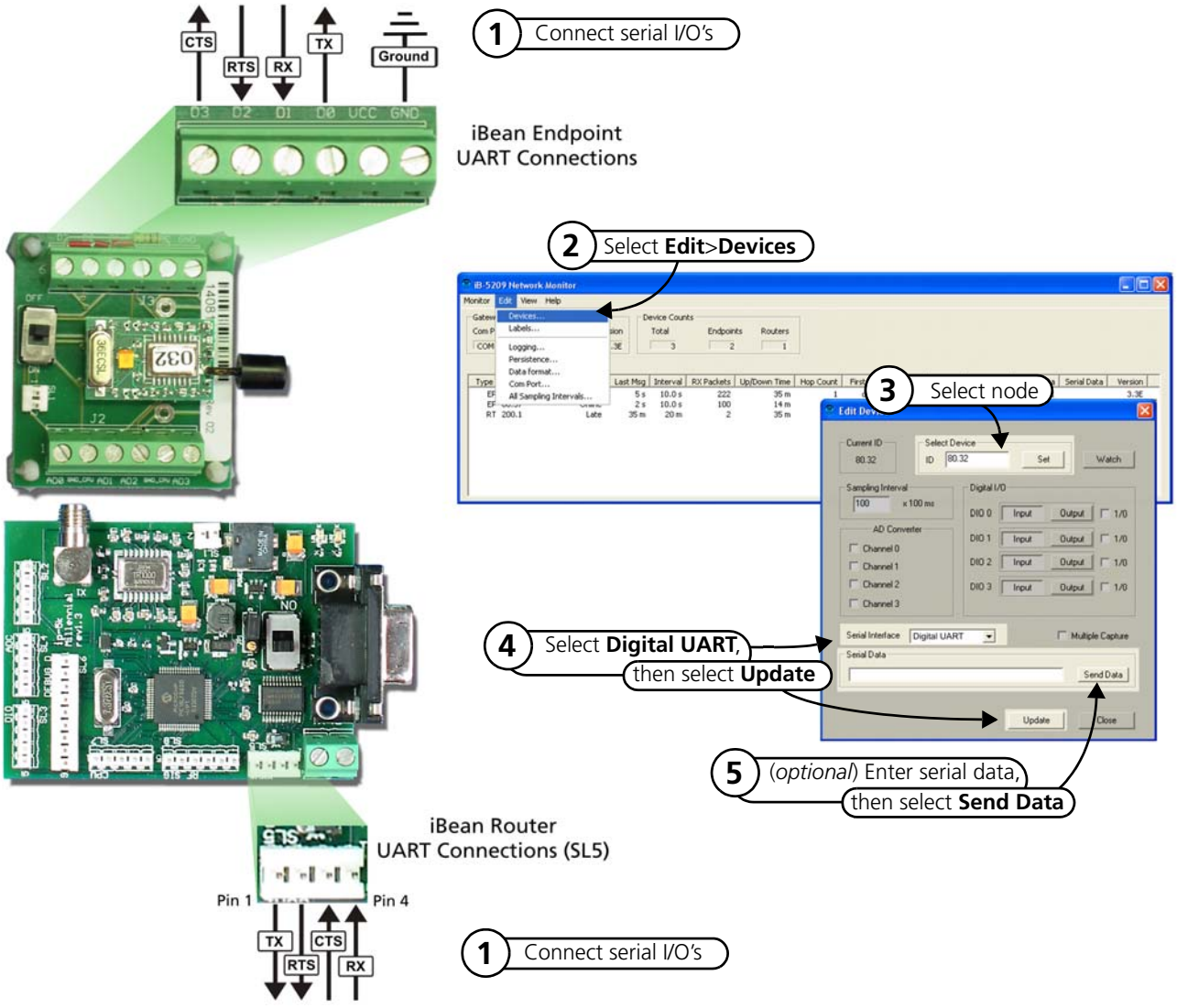
1. Connect to the following:
   - i-Bean Endpoint's serial terminal block (pins D0–D3, GND).
   - i-Bean Router's SL5 connector.

Caution

When making serial connections to the i-Bean node,
use an adapter to scale the RS-232 voltages to 3 volt digital logic.

2. From iB-5209 Network Monitor, select **Edit**>**Devices**. The Edit Device window displays.
3. Enter the device ID of the desired node and click **Set**. The Edit Device window is now ready to make any changes to the selected node.
4. From the *Serial Interfaces* panel, select Serial Data: **Digital UART**, then **Update**. The UART terminal block is ready for UART operation (digital function is disabled).
5. (*optional*) To send serial data to the node, enter the data in the *Serial Data* panel, then select **Send Data**.

Figure 3-7. Configuring i-Bean Endpoint/i-Bean Router for UART



Configuring AD (analog-to-digital) Converter Operation

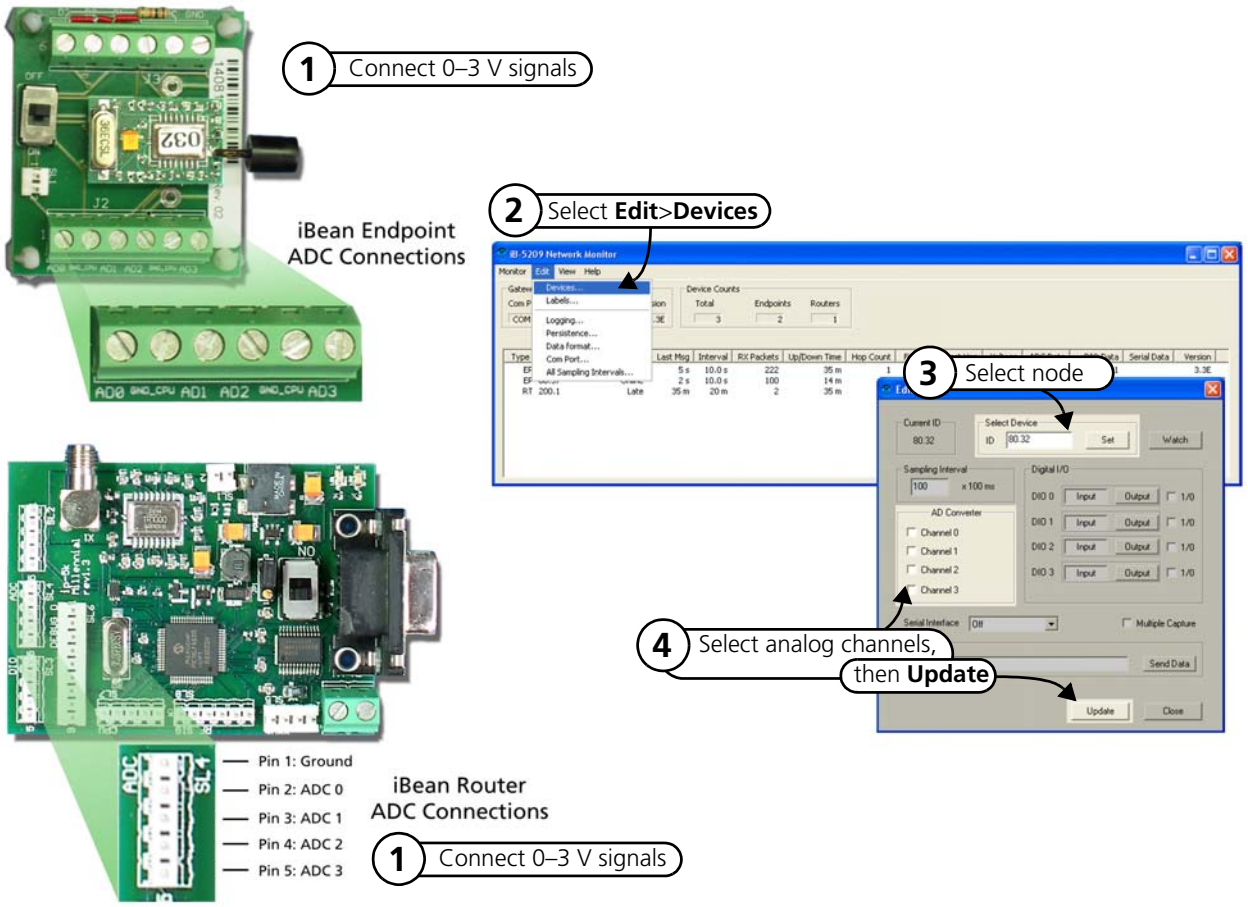
The following procedure describes the steps that need to be taken to set up the hardware and configure an i-Bean Endpoint or i-Bean Router for AD Converter operation (see Figure 3-8):

1. Connect 0–3 VDC signals to the following A/D Converter connectors:
   - i-Bean Endpoint: Connect signal source ground to GND_CPU of the A/D Converter terminal block.
   - i-Bean Router: Connect signal source ground to Pin 1 (Ground) of connector SL4.

   Use any or all of the A/D Converter channels (AD0–AD3/Pins 2–5) for connecting analog signal sources.
2. From iB-5209 Network Monitor, select **Edit**>**Devices**. The Edit Device window displays.
3. Enter the device ID of the desired node and click **Set**. The Edit Device window is now ready to make any changes to the selected node.
4. From the *AD Converter* panel, select the AD Converter channels to which the external analog signals have been connected, then select **Update**.

   iB-5209 Network Monitor displays the analog information for the node in the ADC (V) column.

Figure 3-8. Configuring i-Bean Endpoint/i-Bean Router for analog I/O

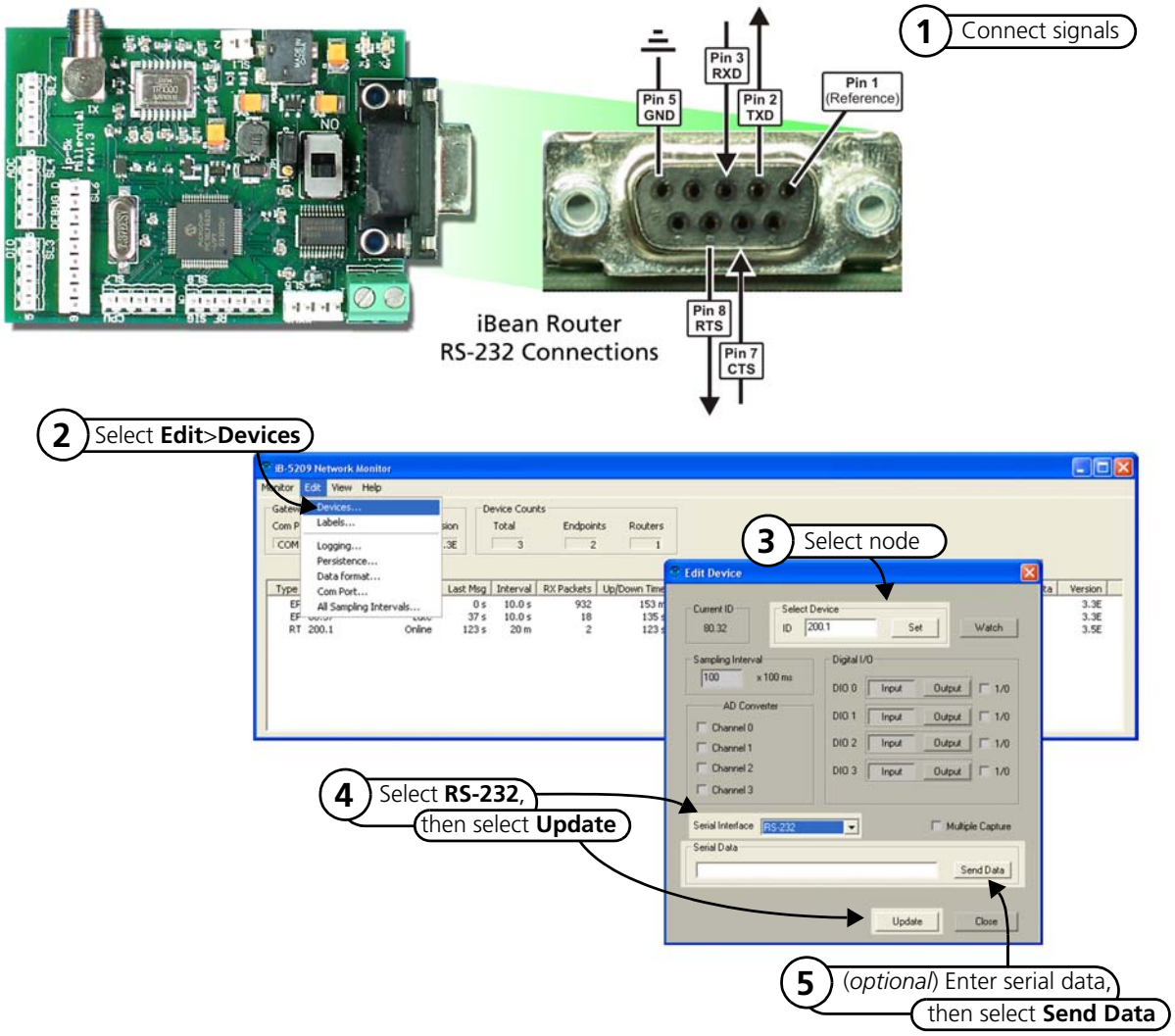


Configuring RS-232 Operation (RT-5209 only)

The following procedure describes the steps that need to be taken to set up the hardware and configure an i-Bean Router for RS-232 operation (see Figure 3-9):

1. Use a DB-9 cable to connect the network device to the RS-232 female connector using the pinout information provided in Figure 3-9.
2. From iB-5209 Network Monitor, select **Edit**>**Devices**. The Edit Device window displays.
3. Enter the device ID of the desired i-Bean Router and click **Set**. The Edit Device window is now ready to make any changes to the selected i-Bean Router.
4. From the *Digital I/O* panel, select Serial Data: **RS-232**, then **Update**. The RS-232 connector is ready for operation (digital function is disabled).
5. (*optional*) To send serial data to the i-Bean Router, enter the data in the *Serial Data* panel, then select **Send Data**.

Figure 3-9. Configuring i-Bean Router for RS-232



Configuring RS-485 Operation (RT-5209 only)

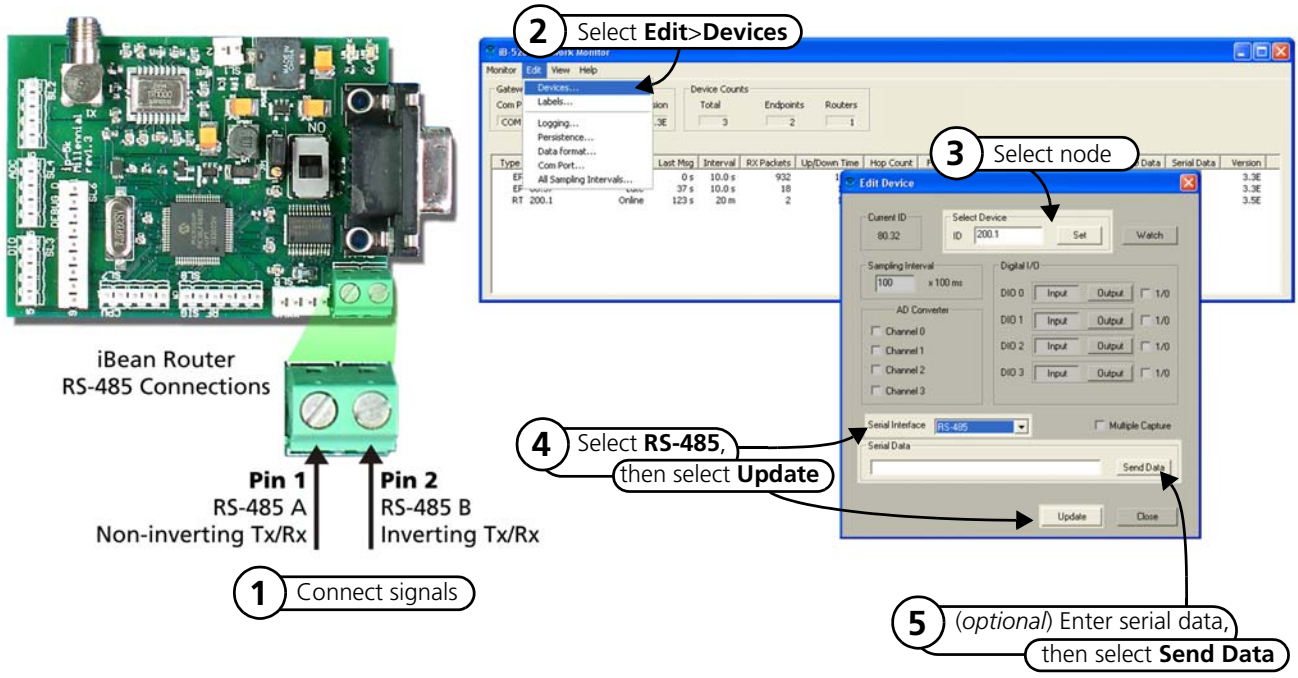
The RS-485 interface of the i-Bean Router operates using the following parameters:

- Half-duplex mode
- 9600 baud
- Transmitting data: Maximum byte length for one request is 80 bytes
- Receiving data: Reception is terminated by the router when the maximum input byte length is reached or there is no input for more than 20 ms. Once the input is terminated, the data is transferred to iB-5209 Network Monitor.

The following procedure describes the steps that need to be taken to set up the hardware and configure an i-Bean Router for RS-485 operation (see Figure 3-10):

1. Connect the network device to the standard 2-position RS-485 terminal block using the pinout information provided in Figure 3-10. The terminal block will accept 26 AWG to 16 AWG wire.
2. From iB-5209 Network Monitor, select **Edit**>**Devices**. The Edit Device window displays.
3. Enter the device ID of the desired i-Bean Router and click **Set**. The Edit Device window is now ready to make any changes to the selected i-Bean Router.
4. From the *Digital I/O* panel, select Serial Data: **RS-485**, then **Update**. The RS-485 connector is ready for operation (digital function is disabled).
5. (*optional*) To send serial data to the i-Bean Router, enter the data in the *Serial Data* panel, then select **Send Data**.

Figure 3-10. Configuring i-Bean Router for RS-485

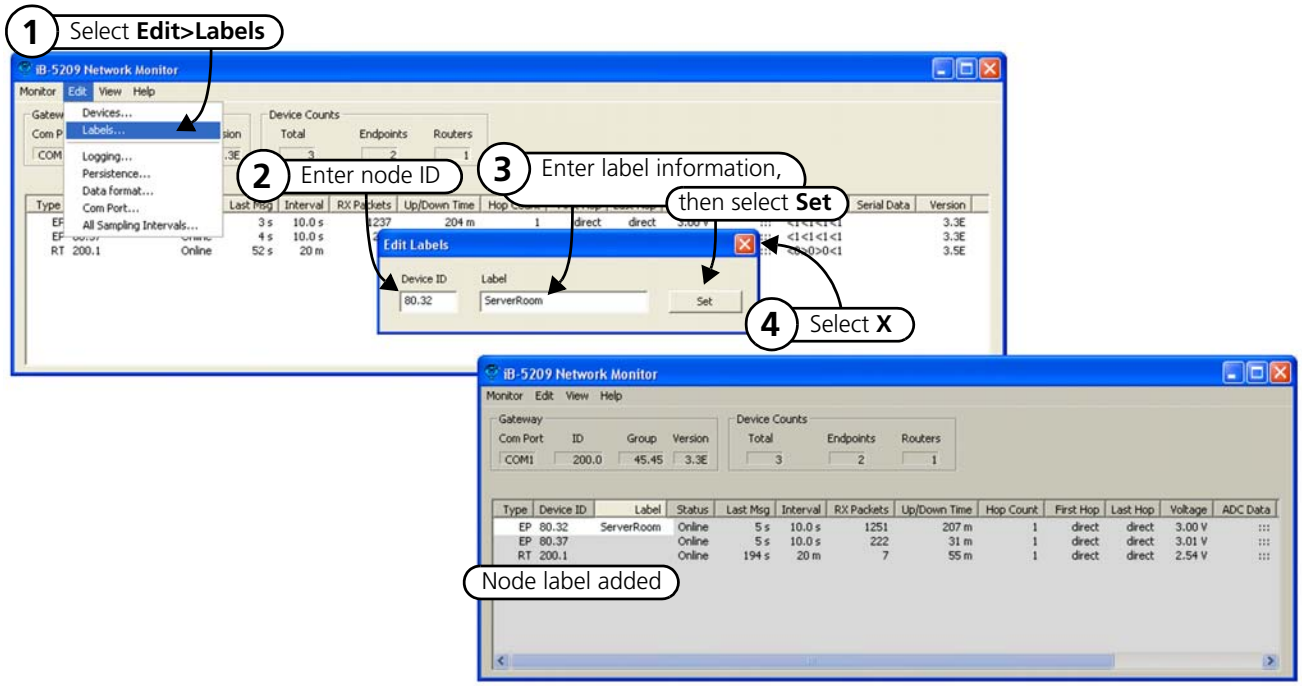


Labeling i-Bean Endpoint or i-Bean Router

The following procedure describes the steps required to assign a label to an i-Bean Endpoint or i-Bean Router (see Figure 3-11):

1. From iB-5209 Network Monitor, select **Edit**>**Labels**. The Edit Labels window displays.
2. Enter the **Device ID** of the desired i-Bean Endpoint or i-Bean Router.
3. Enter the label to be applied to the node(s), then select **Set**. iB-5209 Network Monitor displays the label in the *Label* column of the selected node.
4. Repeat Steps 1–3 for each node, then select **X** to exit the Edit Label window.

Figure 3-11. Labeling i-Bean Endpoint or i-Bean Router



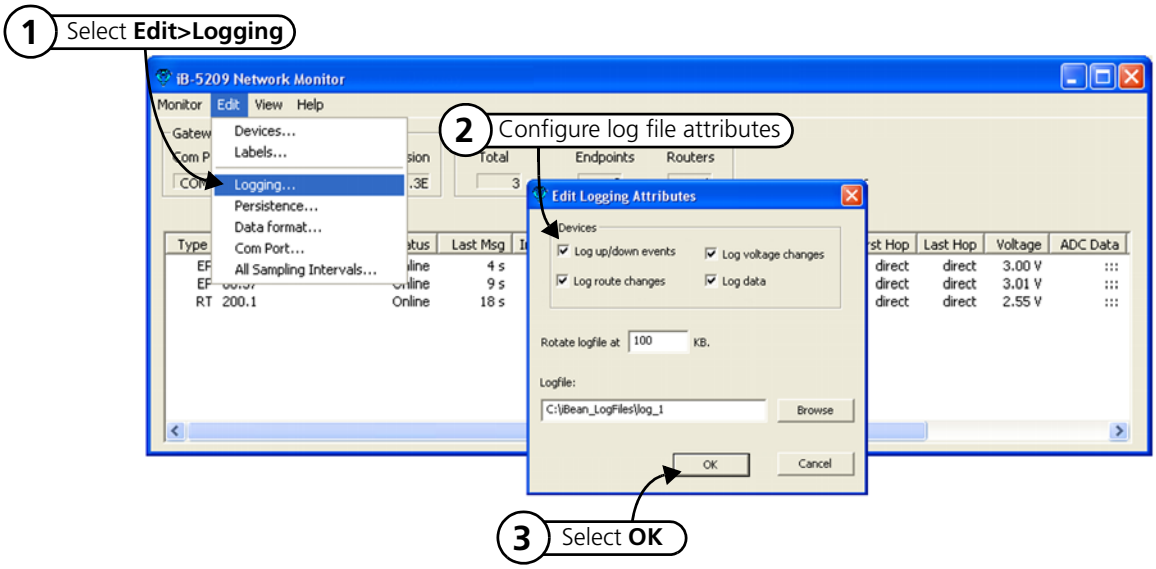
Creating an Event Log File

The following procedure describes the steps required to have iB-5209 Network Monitor record reported network events to a log file (see Figure 3-12):

1. From iB-5209 Network Monitor, select **Edit**>**Logging**. The Edit Logging window displays.
2. Configure the following i-Bean Router and i-Bean Endpoint log file attributes:
   - **Log up/down events**: Record times when a node goes offline/online.
   - **Log route changes**: Record times when a node's hop pattern changes.
   - **Log voltage changes**: Record times when there is a change in a node's power source voltage level.
   - **Log data**: Record serial/ADC/DIO data.
   - **Rotate logfile at *n* KB**: Clear the log file and begin the recording process again when it reaches the designated file size.
   - **Logfile**: Assign a name to the log file and define where the file is saved.
3. Select **OK** to activate the recording process.

To view the contents of the log file, see 'View Contents of Event Log File' on page 3-26.

Figure 3-12. Configure an event log file

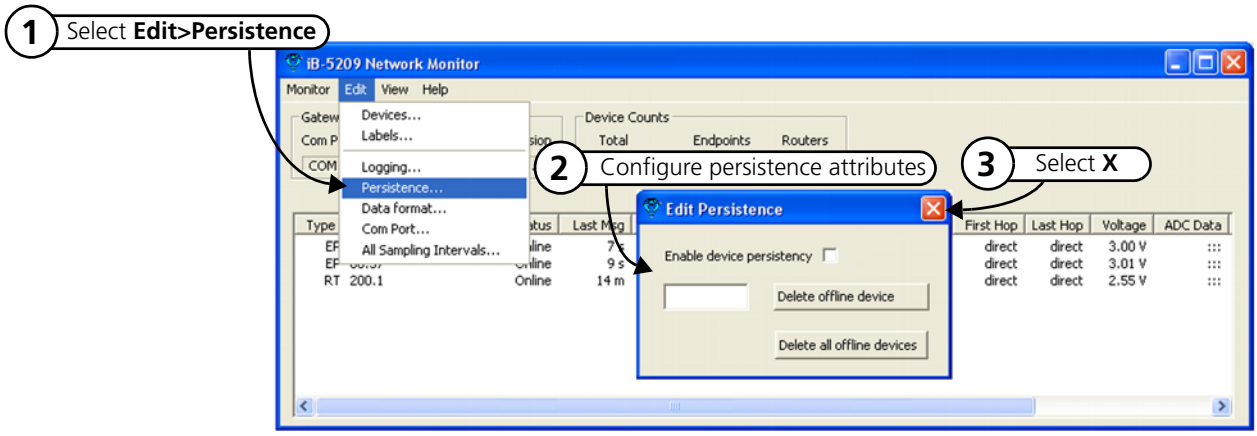


Configure Persistence Attributes

The following procedure describes how to configure the persistence setting of the network's i-Bean Endpoints and i-Bean Routers (see Figure 3-13):

1. From iB-5209 Network Monitor, select **Edit**>**Pesistence**. The Edit Persistence window displays.
2. Configure the following persistence attributes:
   - **Enable device persistency**: If selected, all nodes will be monitored and displayed even if they go offline.
   - **Delete offline device**: Enter a node device ID, then select this delete button to stop monitoring or displaying the selected node.
   - **Delete all offline devices**: Select this delete button to stop displaying any node with a status of Offline.
3. Select **X** to exit the Edit Persistence window.

Figure 3-13. Configuring node persistence attributes

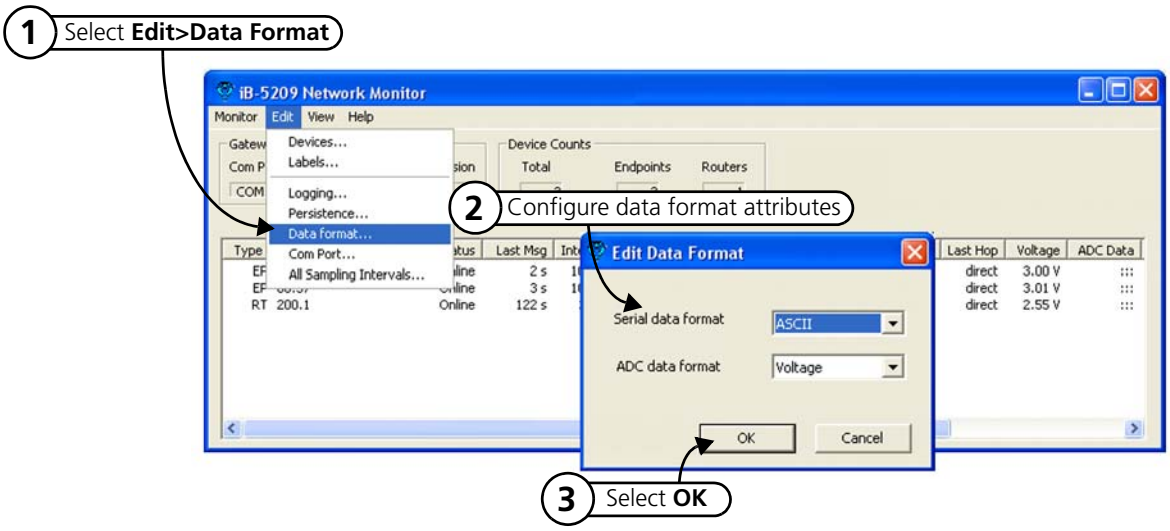


Configure Serial and ADC Data Formats

The following procedure describes how to configure the format of displayed serial and ADC information (see Figure 3-14):

1. From iB-5209 Network Monitor, select **Edit**>**Data format**. The Edit Data Format window displays.
2. Configure the following network attributes:
   - **Serial data format**: Select the desired format for displaying serial data (ASCII/Hex/Decimal).
   - **ADC data format**: Select the desired format for displaying ADC data (Voltage/Raw Data).
3. Select **OK** to save the settings and exit the Edit Network window.

Figure 3-14. Configuring display attributes of iB-5209 Network Monitor



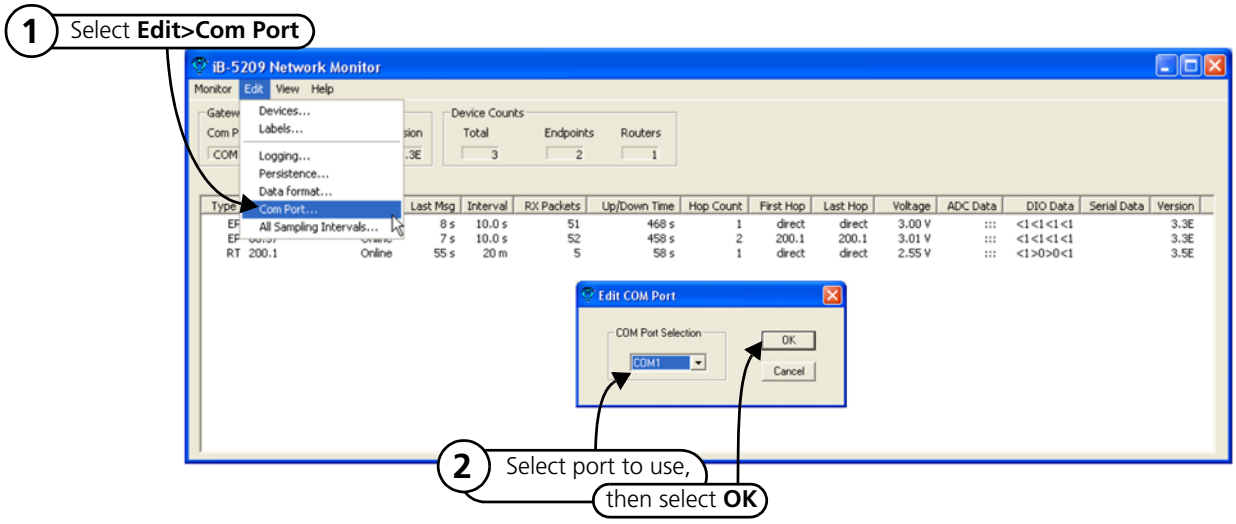
Select Com Port on Host PC

The following procedure describes how to select the RS-232 com port for iB-5209 Network Monitor to use to communicate with the i-Bean Gateway (see Figure 3-15):

1. From iB-5209 Network Monitor, select **Edit**>**Com Port**. The Edit COM Port window displays.

2. Select the RS-232 com port to use from the drop down list of available ports, then select **OK**. iB-5209 Network Monitor will communicate with the i-Bean Gateway connected to the selected port.

Figure 3-15. Selecting com port on host PC

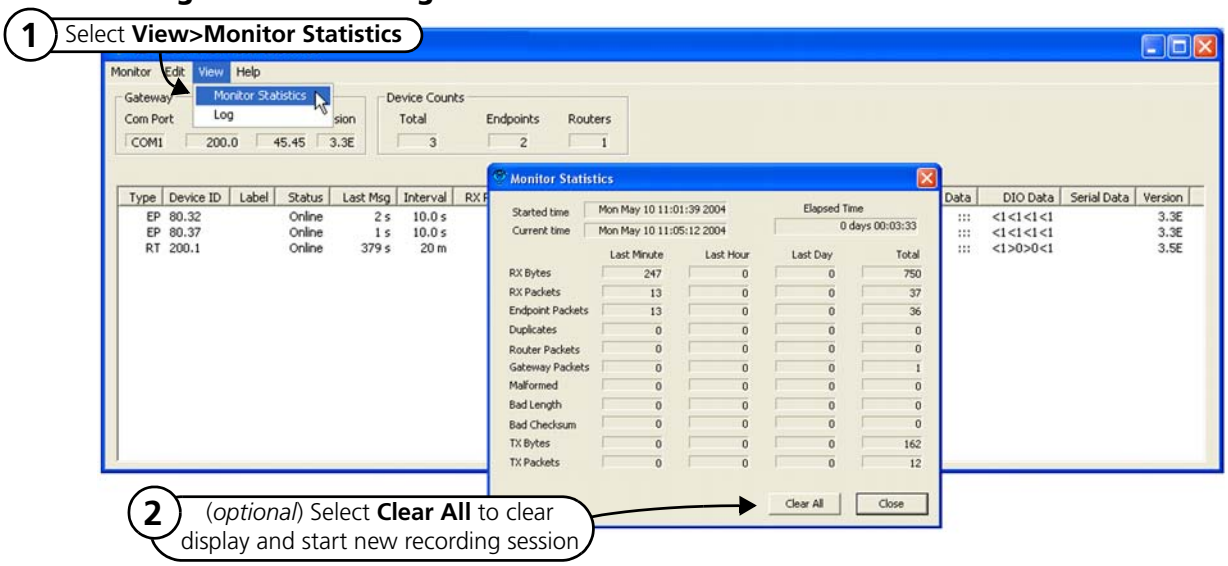


View Monitor Statistics

The following procedure describes how to view the network statistics recorded by iB-5209 Network Monitor, which provides information on the packets received and transmitted by the network nodes (see Figure 3-16):

1. From iB-5209 Network Monitor, select **View**>**Monitor Statistics**. The Monitor Statistics window opens, displaying the following information:
   - **Started Time**: Date/time statistics recording session began.
   - **Current Time**: Current date and time.
   - **Elapsed Time**: Total recording time.
   - **RX Bytes**: Total bytes received.
   - **RX Packets**: Total packets received.
   - **Endpoint Packets**: Packets received from i-Bean Endpoints.
   - **Duplicates**: Duplicate packets received.
   - **Router Packets**: Packets received from i-Bean Routers.
   - **Gateway Packets**: Packets received from i-Bean Gateway.
   - **Malformed**: Packets detected that are malformed.
   - **Bad Length**: Packets detected with incorrect length.
   - **Bad Checksum**: Packets detected with incorrect check sum.
   - **TX Bytes**: Total bytes transmitted.
   - **TX Packets**: Total packets transmitted.
2. (*optional*) Select **Clear All** to clear the information currently displayed and begin a new recording session.

Figure 3-16. Viewing monitored statistics



View Contents of Event Log File

The following procedure describes how to view the network event log file generated by iB-5209 Network Monitor, which records node-generated events (see Figure 3-17):

1. From iB-5209 Network Monitor, select **View**>**Log**. The log file window displays the user-defined event log information (see also 'Creating an Event Log File' on page 3-21).
2. Select one of the following:
   - **Stop/Restart**: Select this toggle option to stop and restart a recording session.
   - **Clear**: Select this option to clear the information currently displayed and start a new recording session.
   - **Close**: Exit and close the log file display window.

To configure the attributes of the log file that get recorded and displayed, see 'Creating an Event Log File' on page 3-21.

Figure 3-17. View contents of event log file

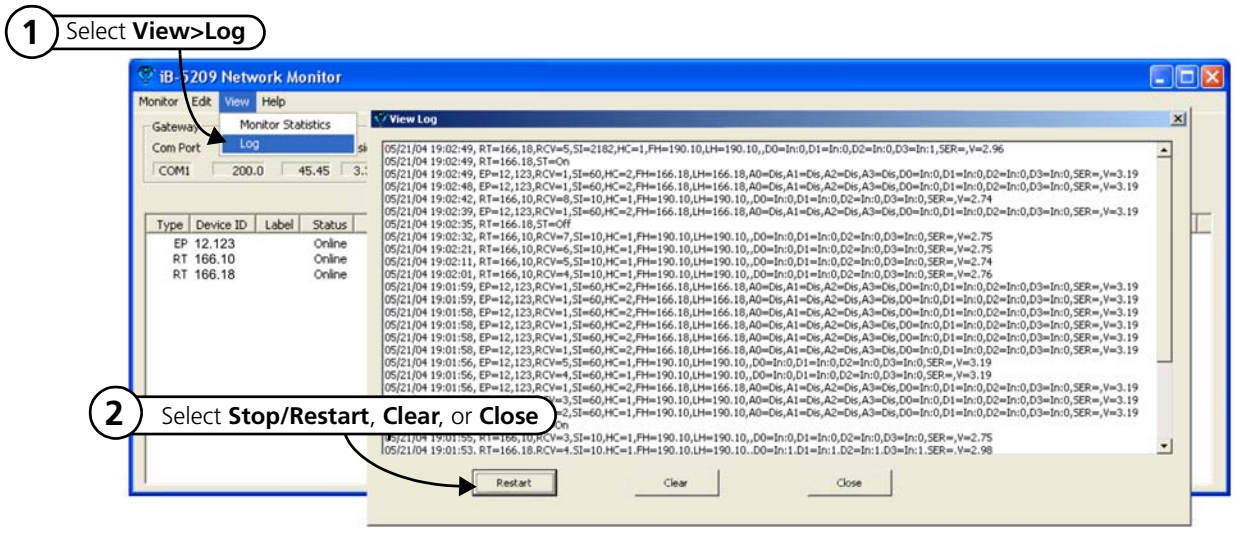


Table 3-3 describes the event keys (see example) displayed in the log file.

example: `04/22 11:00:42.123, EP=001.001 RCV=100, SI=100.0, FH=123.123, LH=123.123 HC=3, ST=On, A0=1.23, D0=In:2`

Table 3-3. Event Log Key Definitions

Key	Key Meaning	Sub-Key	Sub-Key Meaning	Output to device example	Input to device example
A0 ... A3	ADC Channel 0 ... 3	En	Enabled	A0=En	A0=2.12
		Dis	Disabled	A1=Dis	
D0 ... D3	Digital I/O Channel 0 ... 3	In	Input	D0=In	D0=In:1
		Out	Output	D1=Out:1	

Table 3-3. Event Log Key Definitions (continued)

Key	Key Meaning	Sub-Key	Sub-Key Meaning	Output to device example	Input to device example
EP	Endpoint	N/A	N/A	EP=001.001	EP=001.002
FH	First Hop Router	N/A	N/A	N/A	FH=001.002
HC	Hop COunt	N/A	N/A	N/A	HC=3
LH	Last Hop Router	N/A	N/A	N/A	LH=002.003
RCV	Receive	N/A	N/A	N/A	RCV=200 (sequence number)
RT	Router	N/A	N/A	RT=200.001	RT=200.001
SER	Serial Interface	N/A	N/A	SER=12 34 56 13	SER=12 34 56 13
SI	Sampling Interval	N/A	N/A	SI=100.0	SI=200.0
SND	Send	N/A	N/A	SND=100 (sequence number)	N/A
ST	State	On	Online	ST=On	
		Off	Offline		
		Q	Queuing		
		R	Refreshing		

Enable Multiple Capture

Multiple capture is a special feature that enables iB-5209 Network Monitor to report the surrounding router information of a given device (an i-Bean Endpoint or a i-Bean Router that has endpoint functionality). When the Multiple Capture function of a device is enabled, all i-Bean Routers within direct communication distance to that device will report their existence to iB-5209 Network Monitor. The information will be presented as a list of First Hop routers of the given device. The first one on the list is the primary First Hop router, and the rest (if any) are the surrounding routers.

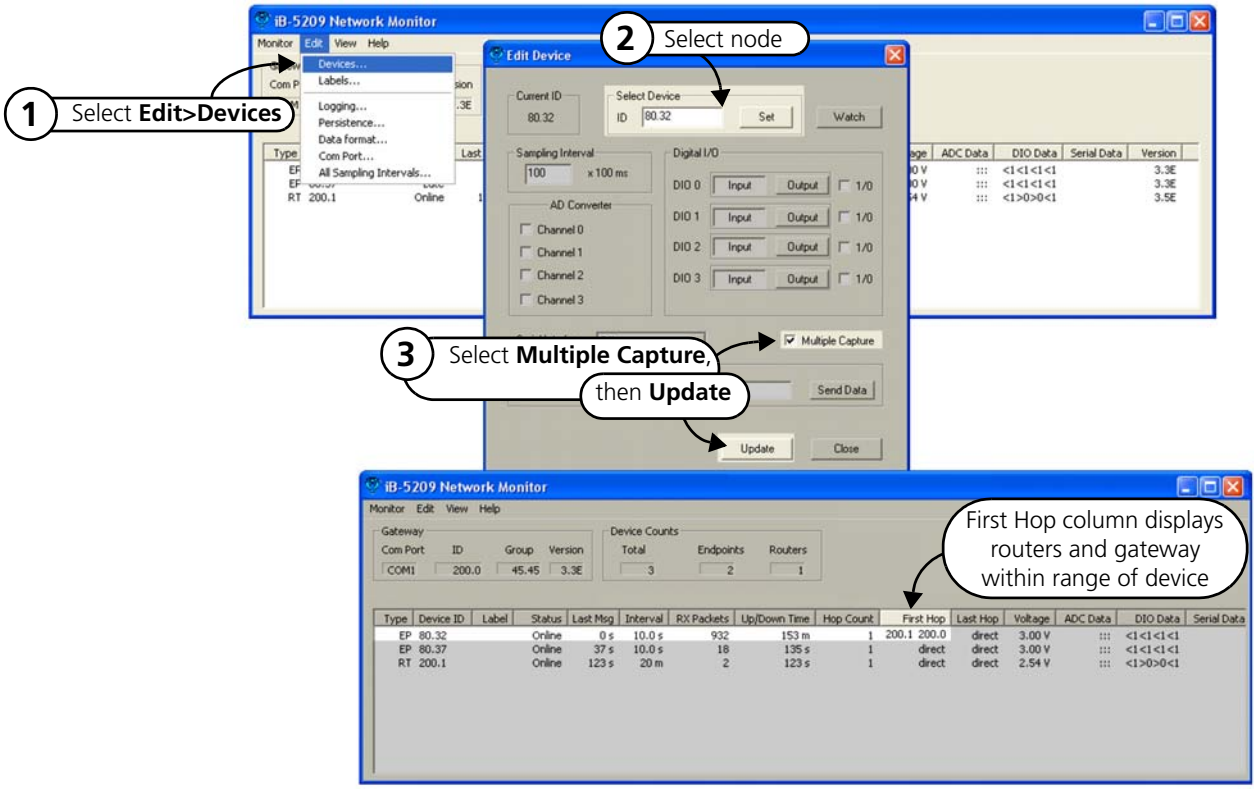
By using the Multiple Capture information displayed, one can infer from the known location of routers the rough location of the given device.

Note: The Multiple Capture function reports extra information to iB-5209 Network Monitor, thereby increasing network traffic. Consider network traffic volume when enabling this function, especially if several devices have this function enabled.

The following procedure describes how to enable Multiple Capture (see Figure 3-18):

1. From iB-5209 Network Monitor, select **Edit**>**Devices**. The Edit Device window displays.
2. Enter the device ID of the desired device and click **Set**. The Edit Device window is now ready to make any changes to the selected device.
3. Select **Multiple Capture**, then **Update**. All i-Bean Routers within range of the device are displayed in the First Hop column of iB-5209 Network Monitor.

Figure 3-18. Enable Multiple Capture



A

API Functions

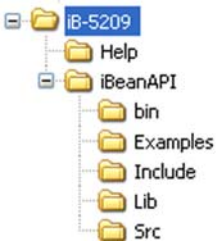
This chapter describes the following API functions:

- 'iB-5209 API Overview' on page A-2
- 'i-Bean API Functions Overview' on page A-4
- 'iBeanAPI.h' on page A-5
- 'iBeanAPI_IO.h' on page A-16
- 'iBeanAPI_Utils.h' on page A-24
- 'Example API Code' on page A-27

iB-5209 API Overview

The iBeanAPI sub-directory contains five sub-directories which hold the API related files, including header files, dll and library files, and various compiled examples along with their source code listings. Figure A-1 is a representation of the directory structure.

Figure A-1. iBeanAPI directories



The contents of each directory is described below.

bin Directory:

This sub-directory contains iMDLL5k.dll file required for running API related applications. This dll file is compiled with Microsoft Visual Studio .Net edition therefore only supports Microsoft Visual C++ programming conventions.

Examples Directory:

This sub-directory contains iMDLL5k.dll file along with pre-compiled API example executables. These executables are compiled with Microsoft Visual Studio .Net edition and are designed to run under Windows XP and Windows 2000. The source code along with sample Microsoft Visual Studio .Net solution file for these example applications can be found in the iBeanAPI/Src sub-directory. Here is a brief description of the API example applications:

- ListDevicesVC7.exe: This is a console based application which lists current online devices. For a detailed look at this example and its code, see 'Example API Code' on page A-27.
- ReadSerialVC7.exe: This is a console based application which reads serial data received from any online device on the network that has serial interface enabled.
- SetSamplingVC7.exe: This is a console based application which changes the sampling interval of the online devices.

To access the different API examples available in the Examples directory, select the following:

Start>All Programs>iB-5209>API Examples

Include Directory:

This sub-directory contains the iBeanAPI header files required to build any API based application.

Lib Directory:

This sub-directory contains the iMDLL5k.lib file required to compile any API based applications.

Src Directory:

This sub-directory contains the source code and their corresponding Microsoft Visual Studio .Net solution files. When these examples are build from these solution files the executables are generated in the bin directory.

i-Bean API Functions Overview

Table A-1 provides a list of API functions (version 2.0) associated with i-Bean network products.

Table A-1. i-Bean API functions

iBeanAPI.h		**Core API functions**

	`ibApi_Open()`
	`ibApi_Close`
	`ibApi_GetApiVersion()`
		Session Management

	`ibApi_GetNetworkList()` `ibApi_GetDeviceList()`	Enumeration of network devices
	`ibApi_GetDeviceInfo()` `ibApi_GetDeviceStatus()`	Static and dynamic device attributes

	`iBApi_GetDeviceState()`
	`ibApi_SetSamplingInterval()`
	`ibApi_GetSamplingInterval()`
		Universally supported device properties

	`ibApi_WaitForDeviceEvent()`	Event Notification
iBeanAPI_IO.h		**Standard I/O peripherals**
	`ibApi_IO_GetDeviceCaps()`	Static device attributes

	`ibApi_IO_SetADCConfig()`
	`ibApi_IO_GetADCConfig()`
	`ibApi_IO_ReadADC()`
		Analog-to-Digital conversion

	`ibApi_IO_SetDIOConfig()`
	`ibApi_IO_GetDIOConfig()`
	`ibApi_IO_WriteDIO()`
	`ibApi_IO_ReadDIO()`
		Digital input/output

	`ibApi_IO_SetSerialConfig()`
	`ibApi_IO_GetSerialConfig()`
	`ibApi_IO_GetSerialBufferStatus()`
	`ibApi_IO_WriteSerial()`
	`ibApi_IO_ReadSerial()`
		Serial data interface (UART)

iBeanAPI_Utils.h		**Supplementary functions**
	`ibApi_Utils_GetErrorDescription()`	Obtain text descriptions for error codes

	`ibApi_Utils_ConvertGroupIdToText()`
	`ibApi_Utils_ConvertTextToGroupId()`
	`ibApi_Utils_ConvertDeviceIdToText()`
	`ibApi_Utils_ConvertTextToDeviceId()`
		Convert between ID structures and text representation

iBeanAPI.h

Data Structures

1. **ibApi_APIHANDLE**

```
typedef ibApi_UINT16 ibApi_APIHANDLE;
```

This handle represents an API session. It is created by `ibApi_Open()` and used by most of the other API functions.

2. **ibApi_RESULT**

```
typedef enum ibApi_RESULT_e ibApi_RESULT;
```

The API functions are standardized to return the value ibApi_RESULT, which is a signed 32-bit integer. If the integer is negative, then it is an error code such as ibApi_RESULT_ERR_INVALIDHANDLE or ibApi_RESULT_ERR_NOTPERMITTED. (See iBeanAPI.h for a full listing of error codes.) Otherwise, the result can be ibApi_RESULT_SUCCESS or a non-negative value specific to the particular function.

3. **ibApi_GROUPID**

```
struct ibApi_GROUPID_s {
   ibApi_UINT8 words[4];
};
typedef struct ibApi_GROUPID_s ibApi_GROUPID;
```

The group ID is a 32-bit address that is used to identify a specific network of i-Bean devices and is shared by all the devices within the network. (In the current implementation, each i-Bean network group can only have one i-Bean Gateway.) A range of group IDs is allocated by Millennial Net for each customer.

The API functions are standardized to return the value ibApi_RESULT, which is a signed 32-bit integer. If the integer is negative, then it is an error code such as ibApi_RESULT_ERR_INVALIDHANDLE or ibApi_RESULT_ERR_NOTPERMITTED. (See iBeanAPI.h for a full listing of error codes.) Otherwise, the result can be ibApi_RESULT_SUCCESS or a non-negative value specific to the particular function.

4. **ibApi_DEVICEID**

```
struct ibApi_DEVICEID_s {
   ibApi_UINT8 words[8];
};
typedef struct ibApi_DEVICEID_s ibApi_DEVICEID;
```

The device ID is a 64-bit address that uniquely identifies an i-Bean network component such as endpoint, router, or gateway.

5. **ibApi_COMPARISON**

```
enum ibApi_COMPARISON_e {
   ibApi_COMPARISON_LESSTHAN,
   ibApi_COMPARISON_EQUAL,
   ibApi_COMPARISON_GREATERTHAN
};
typedef enum ibApi_COMPARISON_e ibApi_COMPARISON;
```

This enum is used for the return value of functions that compare things. Note that these values are non-negative to enable casting as ibApi_RESULT.

6. **ibApi_IOMODE**

```
enum ibApi_IOMODE_e {
   ibApi_IOMODE_OUTPUT=0,
   ibApi_IOMODE_INPUT=1
};
typedef enum ibApi_IOMODE_e ibApi_IOMODE;
```

This is used by functions such as `ibApi_IO_SetDIOConfig()` for configuring channels for input or output.

7. **ibApi_DEVICETYPE**

```
enum ibApi_DEVICETYPE_e {
    ibApi_DEVICETYPE_ENDPOINT = (1<<0),
    ibApi_DEVICETYPE_ROUTER = (1<<1),
    ibApi_DEVICETYPE_ROUTERBEAN = (1<<2),
    ibApi_DEVICETYPE_GATEWAY = (1<<3),
    ibApi_DEVICETYPE_ANY = 0xf  /* used with filter */
};
typedef enum ibApi_DEVICETYPE_e ibApi_DEVICETYPE;
```

This is used to identify the device type. The `ibApi_DEVICETYPE_ENDPOINT` and `ibApi_DEVICETYPE_ROUTERBEAN` implement various I/O interfaces, whereas `ibApi_DEVICETYPE_ROUTER` and `ibApi_DEVICETYPE_GATEWAY` do not.

8. **ibApi_DEVICEINFO**

```
#define ibApi_MAX_VERSION_STRLEN 32
struct ibApi_DEVICEINFO_s {
    ibApi_UINT32 struct_size;
    ibApi_DEVICETYPE device_type;
    ibApi_CHAR hardware_version[ibApi_MAX_VERSION_STRLEN];
    ibApi_CHAR firmware_version[ibApi_MAX_VERSION_STRLEN];
};
typedef struct ibApi_DEVICEINFO_s ibApi_DEVICEINFO;
```

This data structure is used by `ibApi_GetDeviceInfo()` to report static device attributes that are fixed at manufacturing time.

Structure Fields:

struct_size	The value sizeof (`ibApi_DEVICEINFO`) should be assigned to this field prior to calling `ibApi_GetDeviceInfo()`. This allows future versions of the API to extend the struct without breaking binary compatibility.
device_type	The type of the device (endpoint, router, etc.).
hardware_version *firmware_version*	These two fields report the firmware and hardware version strings for various network devices, which are useful for diagnostic purposes. An empty string may be assigned if the device does not support version reporting.

9. **`ibApi_DEVICESTATE`**

```
enum ibApi_DEVICESTATE_e {
    ibApi_DEVICESTATE_ONLINE,
    ibApi_DEVICESTATE_OFFLINE,
    ibApi_DEVICESTATE_CONNECTING,
    ibApi_DEVICESTATE_REFRESHING,
    ibApi_DEVICESTATE_LATE
};
typedef enum ibApi_DEVICESTATE_e ibApi_DEVICESTATE;
```

These functions are used with `ibApi_GetDeviceState()`. When a command is issued to modify a network device, a series of network communications must occur before the change will take effect. During this time period the said to be "refreshing", and the actual device state may be different from values visible to the API. The refresh time depends on many factors such as sampling interval, traffic level, network topology, etc.

10. **`ibApi_DEVICESTATUS`**

```
struct ibApi_DEVICESTATUS_s {
   ibApi_UINT32 struct_size;
   ibApi_UINT16 hop_count;
   ibApi_DEVICEID first_hop_router;
   ibApi_DEVICEID last_hop_router;
   ibApi_FLOAT battery_level;
   ibApi_DEVICESTATE state;
   ibApi_GROUPID group_id;
};
typedef struct ibApi_DEVICESTATUS_s ibApi_DEVICESTATUS;
```

This struct is used by `ibApi_FUNC ibApi_GetDeviceStatus()` to report read-only device properties that change with time.

Structure Fields:

struct_size	The value size of (`ibApi_DEVICESTATUS`) should be assigned to this field prior to calling `ibApi_GetDeviceStatus()`. This allows future versions of the API to extend the struct without breaking binary compatibility.
hop_count	The hop count measures a device's topological distance from the gateway. If the device talking directly to the gateway (i.e., no routers), then the hop count is 1.
first_hop_router	These fields store the device ID of the first and last router that the device's packets passed through on their way to the gateway. If the hop count is 1, then these fields are NULL.
last_hop_router	These fields store the device ID of the first and last router that the device's packets passed through on their way to the gateway. If the hop count is 1, then these fields are NULL.
battery_level	This reports the device battery level measured in volts. The precision is device dependent and typically nonlinear. If battery information is unavailable, the value is 0.
state	The device state. See ibApi_DEVICESTATE documentation for details.
group_id	This reports the group ID currently assigned to the device.

11. **ibApi_DEVICEEVENTTYPE**

```
enum ibApi_DEVICEEVENTTYPE_e {
   ibApi_DEVICEEVENTTYPE_ALL = 0xffffffff
};
typedef enum ibApi_DEVICEEVENTTYPE_e
ibApi_DEVICEEVENTTYPE;
```

In a future release, it will be possible to filter the events reported by `ibApi_WaitForDeviceEvent()` using a bitwise "OR" of the event types defined in this enum. For the current release, the parameter should always be `ibApi_DEVICEEVENTTYPE_ALL`.

12. **ibApi_VERSION**

```
typedef ibApi_UINT32 ibApi_VERSION;
#define ibApi_MAKE_VERSION(MAJOR,MINOR,RELEASE)
    ((ibApi_VERSION)((MAJOR<<16)|(MINOR<<8)|RELEASE))
#define ibApi_GET_VERSION_MAJOR(VER)    ((VER>>16) & 0xff)
#define ibApi_GET_VERSION_MINOR(VER)    ((VER>>8) & 0xff)
#define ibApi_GET_VERSION_RELEASE(VER)  (VER & 0xff)
```

The ibApi_VERSION type is used by functions such as `ibApi_GetApiVersion()` to encode version numbers as a 32-bit integer. Binary compatibility is only guaranteed when the major and minor components are the same. Note that this is a non-negative number to enable casting as ibApi_RESULT.

13. **ibApi_EXPECTED_VERSION**

```
#define ibApi_EXPECTED_VERSION
ibApi_MAKE_VERSION(1,0,0)
```

This macro encodes the API version number that the application was compiled with. It is passed to `ibApi_Open()` as a safeguard to ensure that the correct DLL file is being loaded by the application.

Functions

1. **ibApi_Open**

```
ibApi_FUNC ibApi_Open(
   ibApi_VERSION                    expected_version,
   ibApi_CONST ibApi_CHAR *         server_type,
   ibApi_CONST ibApi_CHAR *         connection_str
);
```

`ibApi_Open()` should be called to initialize the API before any other function is called. The "server_type" parameter specifies the type of connection, and connection_str contains various connection parameters that vary according to server type.

Notes: 1. For the current release, the server_type should always be "local", and the connection string should be "".

2. These text strings are case-sensitive.

Parameters:

expected_version: (input) Should always be ibApi_EXPECTED_VERSION.

server_type: (input) Should always be "local", reserved for future use.

connection_str: (input) "", reserved for future use.

Return Value:

An ibApi_APIHANDLE value if successful, error code (<0) if not.

2. **ibApi_Close**

```
ibApi_FUNC ibApi_Close(
   ibApi_APIHANDLE                  api_hdl
```

This disconnects from the server and releases the API resources. This should be called before your application exits to avoid resource leaks.

Parameter:

api_hdl: (input) API handle returned from `ibApi_Open()`

Return Value:

An ibApi_RESULT_SUCCESS if successful, error code (<0) if not.

3. **ibApi_GetApiVersion()**

```
ibApi_FUNC ibApi_GetApiVersion ();
```

This function returns the actual software version for the API, which can differ from ibApi_EXPECTED_VERSION if DLL's are mixed.

Return Value:

An IbApi_VERSION value if successful, error code (<0) if not.

---

4. **ibApi_GetNetworkList()**

```
ibApi_FUNC ibApi_GetNetworkList(
   ibApi_APIHANDLE                  api_hdl,
   ibApi_GROUPID                    networks[],
   ibApi_UINT32                     networks_size
);
```

This retrieves a list of group ID's for the networks managed by the server.

Parameters:

param api_hdl: (input) API handle returned from `ibApi_Open()`

networks: (output) array of group ID's that is managed by the server

networks_size: (input) ibApi_INT32, maximum size for the network[]

Return Value:

The actual number of networks (which can exceed networks_size if the written data was truncated), or an error code (<0) if unsuccessful.

---

5. **ibApi_GetDeviceList()**

```
ibApi_FUNC ibApi_GetDeviceList(
   ibApi_APIHANDLE                  api_hdl,
   ibApi_GROUPID                    network,
   ibApi_DEVICETYPE                 device_type
   ibApi_DEVICEID                   devices[],
   ibAPI_UINT32                     devices_size
);
```

This retrieves the ID's of the devices in the network. The device_type parameter is a bitwise OR of the ibApi_DEVICETYPE constants that filters the result. (To retrieve all devices, use `ibApi_DEVICETYPE_ANY`.)

Parameters:

param api_hdl: (input) API handle returned from `ibApi_Open()`.

network: (input) Group ID of the network.

device_type: (input) Device type filter.

devices: (output) Array of device ID's to store the result.

devices_size: (input) Maximum size for devices[].

Return Value:

The actual number of devices (which can exceed devices_size if the written data was truncated), or an error code (<0) if unsuccessful.

---

6. **`ibApi_GetDeviceInfo()`**

```
ibApi_FUNC ibApi_GetDeviceInfo(
   ibApi_APIHANDLE                  api_hdl,
   ibApi_DEVICEID                   device_id,
   ibApi_DEVICEINFO *               device_info
);
```

This function retrieves various static device attributes that are predetermined at manufacturing time. Thus, these values only need to be queried once for a particular device. See `ibApi_DEVICEINGFO` above for details.

Note: To avoid memory corruption, size of (`ibApi_DECVICEINFO`) must be assigned to the "struct_size" field prior to calling this function.

Parameters:

api_hdl: (input) API handle returned from ibApi_Open().

device_id: (input) ID of device to be accessed.

device_info: (output) Pointer to variable storing the result.

Return Value:

An ibApi_RESULT_SUCCESS if successful, error code (<0) if not.

---

7. **`ibApi_GetDeviceStatus()`**

```
ibApi_FUNC ibApi_GetDeviceStatus(
   ibApi_APIHANDLE                  api_hdl,
   ibApi_DEVICEID                   device_id,
   ibApi_DEVICESTATUS *             device_status
);
```

This function retrieves various read-only device properties whose values can change with time. See `ibApi_DEVICE STATUS` above for details.

Note: To avoid memory corruption, size of (`ibApi_DECVICESTATUS`) must be assigned to the "struct_size" field prior to calling this function.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of device to be accessed.

device_status: (output) Pointer to variable storing the result.

Return Value:

An ibApi_RESULT_SUCCESS if successful, error code (<0) if not.

---

8. **`ibApi_GetDeviceState()`**

```
ibApi_FUNC ibApi_GetDeviceState(
   ibApi_APIHANDLE                  api_hdl,
   ibApi_DEVICEID                   device_id,
);
```

This function queries the current state of a device in the network. See the `ibApi_DEVICESTATE` notes above for details.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of device to be accessed.

Return Value:

An ibApi_DEVICESTATE if successful, error code (<0) if not..

---

9. **`ibApi_SetSamplingInterval()`**

```
ibApi_FUNC ibApi_SetSamplingInterval(
   ibApi_APIHANDLE                  api_hdl,
   ibApi_DEVICEID                   device_id,
   ibApi_UNIT32                     sampling_interval_ms
);
```

This function sets the sampling interval for the device. The sampling interval determines how frequently updates occur; lower values mean quicker response times, at the price of higher bandwidth and power consumption. Typically the current interval must elapse before the new interval will be programmed. When the update has completed, the device state will return from ibApi_DEVISESTATE_REFRESHING to ibApi_DEVICESTATE_ONLINE.

Note: The assigned value will be quantized to the nearest legal value supported by the device, which is typically a multiple of 100 ms larger than 300 ms.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of device to be accessed.

sampling_interval_ms: (input) New sampling interval (in ms).

Return Value:

An ibApi_RESULT_SUCCESS if successful, error code (<0) if not.

---

10. **`ibApi_GetSamplingInterval()`**

```
ibApi_FUNC ibApi_GetSamplingInterval(
   ibApi_APIHANDLE                 api_hdl,
   ibApi_DEVICEID                  device_id,
);
```

This retrieves the sampling interval for the given device, measured in milliseconds. See `ibApi_SetSamplingInterval()` above.

---

11. **`ibApi_WaitForDeviceEvent()`**

```
ibApi_FUNC ibApi_WaitForDeviceEvent(
   ibApi_APIHANDLE                 api_hdl,
   ibApi_DEVICEEVENTTYPE           event_types,
   ibApi_INT32                     timeout ms,
   ibApi_DEVICEID *                device_id
);
```

This function implements the simplest form of event notification using the application thread content: It causes the calling thread to sleep until a network packet has arrived (i.e., the sequence number has incremented), and then returns the ID of the device that was updated. If multiple devices have changed since the last call, `ibApi_WaitForDeviceEvent()` will return their ID's in sequential round-robin order. If time timeout expires and nothing has changed, the return value ibApi_RESULT_ERR_TIMEOUT.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

event_types: (input) This parameter is reserved for a future feature allowing the wait condition to be restricted to a subset of the possible event types. In the current release, the parameter should always be `ibApi_DEVICEEVENTTYPE_ALL`.

timeout_ms: Number of milliseconds to wait before giving up (use -1 to wait indefinitely).

device_id: (output) ID of the device that changed.

Return Value:

An ibApi_RESULT_SUCCESS if a device changed, ibApi_RESULT_ERR_TIMEOUT if not, or an error code (<0) if unsuccessful.

iBeanAPI_IO.h

Data Structures

1. **ibApi_IO_SERIALMODE**

```
enum ibApi_IO_SERIALMODE_e {
    ibApi_IO_SERIALMODE_DISABLED,
    ibApi_IO_SERIALMODE_RS232
};
typedef enum ibApi_IO_SERIALMODE_e ibApi_IO_SERIALMODE;
```

This is used by ibApi_IO_SERIALCONFIG to select the serial interface.

2. **ibApi_IO_SERIALCONFIG**

```
struct ibApi_IO_SERIALCONFIG_s {
    ibApi_UINT32 struct_size;
    ibApi_IO_SERIALMODE mode;
};
typedef struct ibApi_IO_SERIALCONFIG_s ibApi_IO_SERIALCONFIG;
```

Structure Fields:

struct_size — The value size of (`ibApi_IO_SERIALCONFIG`) should be assigned to this field prior to calling `ibApi_IO_SetSerialConfig()`. This allows future versions of the API to extend the struct without breaking binary compatibility.

mode — See `ibApi_IO_SERIALMODE` comments above.

3. **ibApi_IO_DEVICECAPS**

```
struct ibApi_IO_DEVICECAPS_s {
    ibApi_UINT32 struct_size;
    ibApi_UINT8 num_dio_channels;
    ibApi_UINT8 num_adc_channels;
    ibApi_UINT8 adc_resolution_bits;
    ibApi_UINT8 serial_input_buffer_depth:
    ibApi_UINT8 serial_output_buffer_depth
};
typedef struct ibApi_IO_DEVICECAPS_s ibApi_IO_DEVICECAPS;
```

This structure is used by `ibApi_IO_GetDeviceCaps()` to return various static device attributes that are predetermined at manufacturing time.

Structure Fields:

struct_size	The value size of (`ibApi_IO_DEVICECAPS`) should be assigned to this field prior to calling `ibApi_IO_GetDeviceCaps()`. This allows future versions of the API to extend the struct without breaking binary compatibility.
num_dio_channels	This is the number of DIO channels (i.e., the channel index passed to `ibApi_IO_ReadDIO()` must be less than this).
num_adc_channels	This is the number of DIO channels (i.e., the channel index passed to `ibApi_IO_ReadADC()` must be less than this).
adc_resolution_bits	This is the number of bits of resolution supported by the A/D converter, i.e. the maximum value for the raw data will be (1<<adc_rsolution_bits)-1.
serial_input_buffer_depth	This is the number of input data packets slots for which packets can be pending to be read by the API-based application.
serial_output_buffer_depth	This is the number of output data packet slots for which packets can be pending to be send by the monitor.

Functions

1. **`ibApi_IO_GetDeviceCaps()`**

```
ibApi_FUNC ibApi_IO_GetDeviceCaps (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_IO_DEVICECAPS *           device_caps
);
```

This function retrieves various static device attributes that are predetermined at manufacturing time. Thus, these values only need to be queried once for a particular device. See `ibApi_IO_DEVICECAPS` above for details.

Note: To avoid memory corruption, size of (`ibApi_IO_DEVICECAPS`) must be assigned to the "struct_size" field prior to calling this function.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

device_caps: (output) Pointer to variable storing the result.

Return Value:

An ibApi_RESULT_SUCCESS if successful, error code (<0) if not.

2. **ibApi_IO_SetADCConfig()**

```
ibApi_FUNC ibApi_IO_SetADCConfig (
    ibApi_APIHANDLE                    api_hdl,
    ibApi_DEVICEID                     device_id,
    ibApi_UINT8                        channel_index,
    ibApi_BOOL                         enabled
);
```

This sets whether the specified ADC channel is enabled or disabled. The channel must be enabled before calling `ibApi_IO_ReadADC()`.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

channel_index: (input) Channel index to access.

enabled: (input) ibApi_TRUE to enable the channel.

Return Value:

An ibApi_RESULT_SUCCESS if successful, or an error code (<0) if not.

3. **ibApi_IO_GetADCConfig()**

```
ibApi_FUNC ibApi_IO_GetADCConfig (
    ibApi_APIHANDLE                    api_hdl,
    ibApi_DEVICEID                     device_id,
    ibApi_UINT8                        channel_index
);
```

This queries whether the specified ADCchannel is enabled or disabled.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

channel_index: (input) Channel index to access.

Return Value:

An ibApi_TRUE if enabled, ibApi_FALSE if disabled, or an error code (<0) if not.

4. **ibApi_IO_ReadADC()**

```
ibApi_FUNC ibApi_IO_ReadADC (
    ibApi_APIHANDLE                    api_hdl,
    ibApi_DEVICEID                     device_id,
    ibApi_UINT8                        channel_index,
    ibApi_FLOAT *                      adc_value
);
```

This retrieves the value of the specified ADC channel, measured in volts. Note that this is computed by normalizing the raw reading relative to the battery voltage, and this calculation influences the precision of the result.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

channel_index: (input) Channel index to access.

adc_value: (output) Result of the ADC reading in volts.

Return Value:

An ibApi_RESULT_SUCCESSFUL if successful, or an error code (<0) if not.

---

5. **`ibApi_IO_SetDIOConfig()`**

```
ibApi_FUNC ibApi_IO_SetDIOConfig (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_UINT8                     channel_index,
    ibApi_IOMODE                    io_mode
);
```

This sets whether the specified DIO channel is configured for input or output, which governs the interpretation of `ibApi_IO_ReadDIO()` and `ibApi_IO_WriteDIO()`.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

channel_index: (input) Channel index to access.

io_mode: (input) New input/output mode.

Return Value:

An ibApi_RESULT_SUCCESSFUL if successful, or an error code (<0) if not.

---

6. **`ibApi_IO_GetDIOConfig()`**

```
ibApi_FUNC ibApi_IO_GetDIOConfig (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_UINT8                     channel_index
);
```

This queries whether the specified DIO channel is configured for input or output.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

channel_index: (input) Channel index to access.

Return Value:

An ibApi_IOMODE value if successful, or an error code (<0) if not.

---

7. **`ibApi_IO_WriteDIO()`**

```
ibApi_FUNC ibApi_IO_WriteDIO (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_UINT8                     channel_index,
    ibApi_UINT8                     dio_value
);
```

This sets the value of the specified DIO channel. Note that an error will result if the channel is not configured for output.

Note: In some product models, the DIO pins are shared with the serial data interface and will be disabled when it is active.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

channel_index: (input) Channel index to access.

dio_value: (input) New output level (0 or 1).

Return Value:

An ibApi_RESULT_SUCCESS if successful, or an error code (<0) if not.

---

8. **`ibApi_IO_ReadDIO()`**

```
ibApi_FUNC ibApi_IO_ReadDIO (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_UINT8                     channel_index
);
```

This reads the value of the specified DIO channel. If the channel is configured for output, it reads the current output value.

Note: In some product models, the DIO pins are shared with the serial data interface and will be disabled when it is active.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

channel_index: (input) Channel index to access.

Return Value:

Digital 0 or 1, or an error code (<0).

---

9. **`ibApi_IO_SetSerialConfig()`**

```
ibApi_FUNC ibApi_IO_SetSerialConfig (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_IO_SERIALCONFIG *         serial_config
);
```

This configures the serial "user data" interface. If these pins are shared with the DIO pins, the DIO will be disabled.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

serial_config: (input) New configuration.

Return Value:

An ibApi_RESULT_SUCCESS if successful, or an error code (<0) if not.

---

10. **`ibApi_IO_GetSerialConfig()`**

```
ibApi_FUNC ibApi_IO_GetSerialConfig (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_IO_SERIALCONFIG *         serial_config
);
```

This retrieves the configuration for the serial "user data" interface.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

serial_config: (output) Variable to store the result.

Return Value:

An ibApi_RESULT_SUCCESS if successful, or an error code (<0) if not.

---

11. **`ibApi_IO_GetSerialBufferStatus()`**

```
ibApi_FUNC ibApi_IO_GetSerialBufferStatus (
    ibApi_APIHANDLE                          api_hdl,
    ibApi_DEVICEID                           device_id,
    ibApi_IOMODE                             io_mode
);
```

For the given device, this function retrieves the status of the out going serial data buffer. The return value gives the number of empty packet slots in the buffer. A negative return value denotes an error and a zero return value means there is currently no out going empty packet slots.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

io_mode: (input) Data direction to be accessed.

Return Value:

The empty out going packet slots if successful, or an error code (<0) if not.

12. **`ibApi_IO_WriteSerial()`**

```
ibApi_FUNC ibApi_IO_WriteSerial (
    ibApi_APIHANDLE                          api_hdl,
    ibApi_DEVICEID                           device_id,
    ibApi_UINT8                              buffer[],
    ibApi_UINT8                              buffer_size
);
```

This writes a user data packet to the specified device handle. The ibApi_FIELDID_USERDATAMODE field must have been previously set to something other than ibApi_USERDATAMODE_DISABLED. (Note that on some devices, this will disable other peripherals, such as DIO.) The specific contents of the user data block and its maximum size are application defined. The buffer_size variable determines the number of bytes sent (note that NULL bytes receive no special interpretation.) To query the maximum payload size, call `ibApi_WriteSerialData()` with buffer_size=0.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

buffer: (input) User data packet to transmit.

buffer_size: (input) Number of bytes in the user data packet.

Return Value:

The maximum buffer size if successful, or an error code (<0) if not.

13. **`ibApi_IO_ReadSerial()`**

```
ibApi_FUNC ibApi_IO_ReadSerial (
    ibApi_APIHANDLE                 api_hdl,
    ibApi_DEVICEID                  device_id,
    ibApi_UINT8                     buffer[],
    ibApi_UINT8                     buffer_size,
    ibApi_UINT8                     seq_num
);
```

For the given device, this retrieves the user data packet that arrived most recently. The `ibApi_IO_SERIALMODE` setting must have been previously something other than `ibApi_IO_SERIALMODE_DISABLED`. The input buffer holds a single packet (i.e., an arriving packet overwrites the previous one). Lost packets can be detected by gaps in the sequence numbers, which increment whenever a packet is received. If no new data is available, then the return value is 0.

Parameters:

api_hdl: (input) API handle returned from `ibApi_Open()`.

device_id: (input) ID of the device to be accessed.

buffer[ ]: (output) Buffer to store the incoming user data packet.

buffer_size: (input) Maximum size for buffer[ ].

seq_num: (output) Sequence number identifying this packet, or NULL if this information is not needed.

Return Value:

Error code or the actual size of the result (which could exceed buffer_size if the written data was truncated)

iBeanAPI_Utils.h

Functions

1. **`ibApi_Utils_GetErrorDescription()`**

```
ibApi_FUNC ibApi_Utils_GetErrorDescription(
    ibApi_RESULT                    error_code,
    ibApi_CHAR *                    description,
    ibApi_UINT32                    description_size
);
```

This function returns an English language interpretation for an API error code.
Parameters:

error_code: (input) ibApi_RESULT to be interpreted.

description: (output) Pointer to buffer storing the text.

description_size: (input) Maximum size of buffer.

Return Value:

Error code, or the actual size of the result including the terminating NULL (which could exceed description_size if the written data was truncated).

2. **`ibApi_Utils_ConvertGroupIDToText()`**

```
ibApi_FUNC ibApi_Utils_ConvertGroupIDToText(
    ibApi_CONST ibApi_GROUPID       group_id,
    ibApi_CHAR *                    group_id_text,
    ibApi_UINT32                    group_id_text_size,
    ibApi_UINT32                    min_words
);
```

This renders a group ID as a text string, such as "1.2.3.4". If the "min_words" parameter is less than 4,leading zeros will be omitted for brevity. For example, if min_words is 3, then 0.0.5.1 would be rendered as "0.5.1".
Parameters:

group_id: (input) Group ID to convert.

group_id_text: (output) Pointer to buffer storing the text.

group_id_text_size: (input) Maximum size of buffer.

min_words: (input) Minimum number of digit groups.

Return Value:

Error code, or the actual size of the result including the terminating NULL (which could exceed group_id_text_size if the written data was truncated).

3. **ibApi_Utils_ConvertTextToGroupID()**

```
ibApi_FUNC ibApi_Utils_ConvertTextToGroupID(
    ibApi_CONST ibApi_CHAR *        group_id_text,
    ibApi_GROUPID *                 group_id
);
```

This parses a text string such as "1.2.3.4" and stores the result in the group_id structure. If fewer than 4 digit groups are provided, the result is left-padded with 0's.

Parameters:

group_id_text: (input) Buffer to be parsed.

group_id: (output) Structure to store the result.

Return Value:

ibApi_RESULT_SUCCESS if successful, error code (<0) if not.

4. **ibApi_Utils_ConvertDeviceIdToText()**

```
ibApi_FUNC ibApi_Utils_ConvertDeviceIdToText(
    ibApi_DEVICEID                  device_id,
    ibApi_CHAR *                    device_id_text,
    ibApi_UINT32                    device_id_text_size,
    ibApi_UINT32                    min_words
);
```

This renders a device ID as a text string such as "1.2.3.4.5.6.7.8". If the "min_words" parameter is less than 8, leading zeros will be omitted for brevity. For example, if min_words is 4, then 0.0.0.0.0.5.1 would be rendered as "0.0.5.1".

Parameters:

device_id: (input) Device ID to convert.

device_id_text: (output) Buffer to store the text.

device_id_text_size: (input) Maximum size of the buffer.

min_words: (input) Minimum number of digit groups.

Return Value:

Error code, or the actual size of the result including the terminating NULL (which could exceed device_id_text_size if the written data was truncated).

5. **ibApi_Utils_ConvertTextToDeviceId()**

```
ibApi_FUNC ibApi_Utils_ConvertTextToDeviceId(
    ibApi_CONST ibApi_CHAR *        device_id_text,
    ibApi_DEVICEID *                device_id
);
```

This parses a text string such as "1.2.3.4.5.6.7.8" and stores the result in the device_id structure. If fewer than 8 digit groups are provided, the result is left-padded with 0's.

Parameters:

group_id_text: (input) Buffer to be parsed.

group_id: (output) Structure to store the result.

Return Value:

ibApi_RESULT_SUCCESS if successful, error code (<0) if not.

Example API Code

Millennial Net provides as part of the evaluation kit, several example API applications. This section examines in detail one of the API examples, ListDevicesVC7, including the code it uses. ListDevicesVC7 provides a list of all detected network nodes (i-Bean Gateway, i-Bean Router(s), and i-Bean Endpoint(s)).

ListDevicesVC7 Example

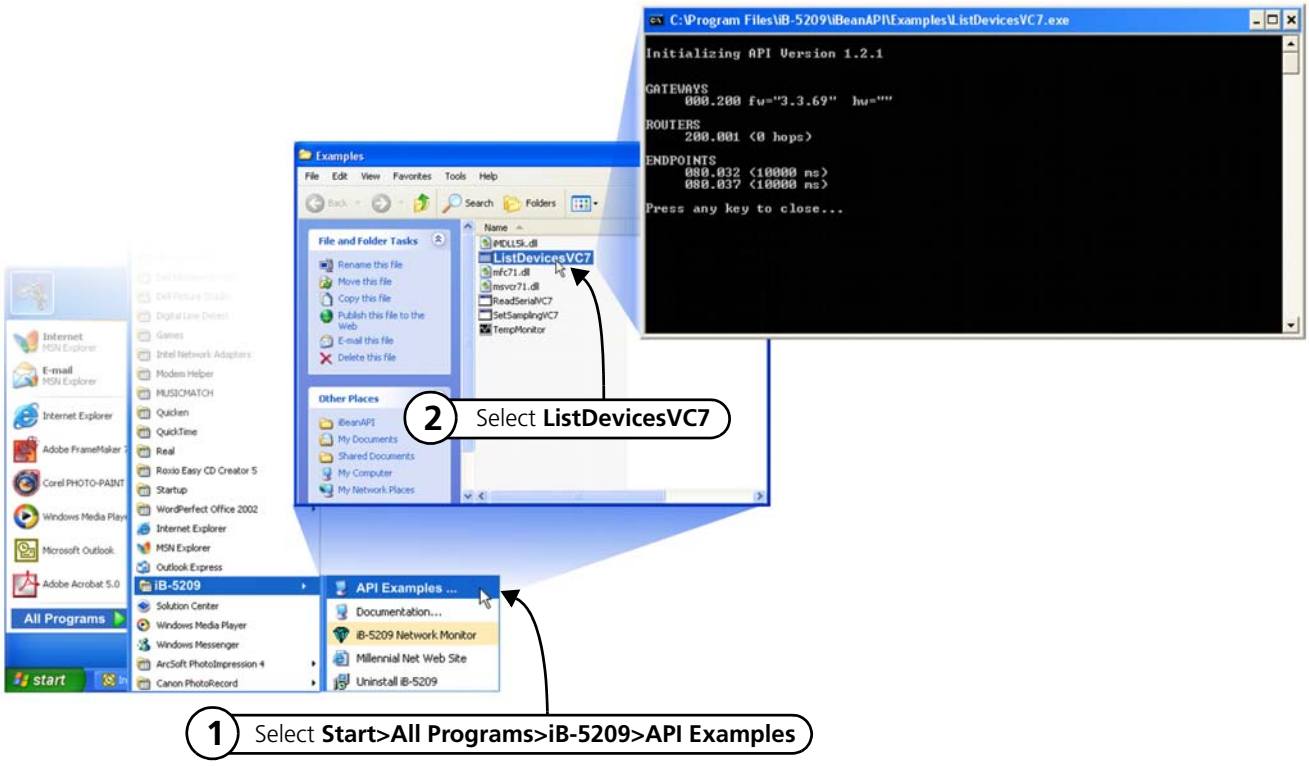
Note: Be sure that iB-5209 Network Monitor is running before executing the following procedure (see 'Launching iB-5209 Network Monitor' on page 2-12).

To run the ListDevicesVC7 example (see Figure 2.):

1. From the Windows taskbar, select the following to open the Examples window: **Start>All Programs>iB-5209>API Examples**
2. From the Examples window, select **ListDevicesVC7**. The command window opens and displays the list of detected network nodes.

Proceed to "ListDevicesVC7 Code" to view the code used in this example.

Figure A-2. API Example: ListDevicesVC7



ListDevicesVC7 Code

The C file containing the code shown here can be found in the Programs directory: Programs\iB-5209\iBeanAPI\Src\ListDevices

```
/*
 * ListDevices.c
 *
 * Copyright (c) 2000-2004 Millennial Net, Inc. All Rights Reserved.
 * Reproduction or modification is strictly prohibited without express
 * written consent of Millennial Net.
 *
 * This example illustrates the basic operations of connecting to the i-Bean
 * API and obtaining basic information about the devices in the network.
 * It prints out the list of gateways, routers, and endpoints currently
 * participating in the network, along with some information about each
 * device.
 *
 * This project was built using Microsoft Visual C++ version 7.1, but should
 * be compatible with other similar compiler versions.
 */

#include <iBeanAPI.h>
#include <iBeanAPI_IO.h>
#include <iBeanAPI_Utils.h>
#include <stdio.h>
#include <stdlib.h>

#ifndef __GNUC__
#include <conio.h>
#else
#include <mingw/conio.h>
/*  #include <mingw/conio.h>  */
#endif

/*
 * This is the number of "words" in a network address.  For example, the
 * address "127.0.1" contains three words.  The i-Bean protocol supports
 * up to 8 words (64-bits), but the actual maximum is reduced in some
 * product releases to optimize the packet size.
 */
#define MIN_DEVICEID_WORDS 3
```

```
/*****************************************************************************/
void WaitForKey(void) {
  printf("\r\nPress any key to close...");
  _getch();
  printf("\r\n");
}

/*****************************************************************************
 * This is a simple wrapper for detecting and reporting API error return
 * values.  In C++, this function could throw an exception object.
 */
ibApi_RESULT CheckResult(ibApi_RESULT result) {
  char error_text[256];

  /*
   * Error codes always have a negative value.
   */
  if (result >= 0) return result;

  /*
   * For the purposes of this example, ibApi_RESULT_ERR_TIMEOUT is not a
   * fatal error.
   */
  if (result == ibApi_RESULT_ERR_TIMEOUT) return result;

  /*
   * This interprets the error code, writing the result to the error_text
   * variable
   */
  ibApi_Utils_GetErrorDescription(result,error_text,sizeof(error_text));

  printf("\r\nERROR: %s\r\n",error_text);

  /*
   * Technically, ibApi_Close() should be called before exiting, e.g. via
   * an atexit() handler.  (This is omitted in the example for simplicity.)
   */
  WaitForKey();

  exit(1);

  return 0;
}
```

```
/*****************************************************************************/
void ListDevices(ibApi_APIHANDLE api_hdl) {
#define DEVICEIDS_MAX 100
  ibApi_GROUPID groupid;
  ibApi_DEVICEID deviceids[DEVICEIDS_MAX];
  int deviceids_count;
  char deviceid_text[256];
  int sampling_interval;
  int i;
  ibApi_DEVICEINFO deviceinfo;
  ibApi_DEVICESTATUS devicestatus;

  /*
   * The ibApi_GetNetworkList() function returns a list of the groups
   * currently managed by the network.  If the gateway is not properly
   * connected to the monitor, then this list will be empty.
   */
  if (CheckResult(ibApi_GetNetworkList(api_hdl,&groupid,1) < 1)) {
    printf("The network is empty\r\n");
    return;
  }

  /*
   * List the gateways in the group, which typically should be
   * only one.
   */
  printf("\r\nGATEWAYS\r\n");
  deviceids_count = CheckResult(ibApi_GetDeviceList(api_hdl, groupid,
    ibApi_DEVICETYPE_GATEWAY, deviceids,DEVICEIDS_MAX));

  /*
   * If the buffer limit was exceeded, then display partial results
   */
  if (deviceids_count > DEVICEIDS_MAX)
    deviceids_count = DEVICEIDS_MAX;

  for (i=0; i<deviceids_count; ++i) {

    /*
     * Note that the struct_size must be assigned BEFORE calling
     * ibApi_GetDeviceInfo().  This allows compatibility with future API
     * versions that implement additional fields.
     */
    deviceinfo.struct_size = sizeof(deviceinfo);
```

```
  CheckResult(ibApi_GetDeviceInfo(api_hdl,deviceids[i],&deviceinfo));

  CheckResult(ibApi_Utils_ConvertDeviceIdToText(deviceids[i],
    deviceid_text,sizeof(deviceid_text),MIN_DEVICEID_WORDS));

  printf("  %10s fw=\"%s\"  hw=\"%s\"\r\n", deviceid_text,
    deviceinfo.firmware_version,deviceinfo.hardware_version);
}

/*
 * List the routers.
 */
printf("\r\nROUTERS\r\n");
deviceids_count = CheckResult(ibApi_GetDeviceList(api_hdl, groupid,
  ibApi_DEVICETYPE_ROUTER|ibApi_DEVICETYPE_ROUTERBEAN, deviceids,DEVICEIDS_MAX));

if (deviceids_count > DEVICEIDS_MAX)
  deviceids_count = DEVICEIDS_MAX;

for (i=0; i<deviceids_count; ++i) {
  /*
   * Note that the struct_size must be assigned BEFORE calling
   * ibApi_GetDeviceStatus().
   */
  devicestatus.struct_size = sizeof(devicestatus);
  CheckResult(ibApi_GetDeviceStatus(api_hdl,deviceids[i],&devicestatus));

  CheckResult(ibApi_Utils_ConvertDeviceIdToText(deviceids[i],
    deviceid_text,sizeof(deviceid_text),MIN_DEVICEID_WORDS));

  printf("  %10s (%i hops)\r\n", deviceid_text, devicestatus.hop_count);
}

/*
 * List the endpoints.
 */
printf("\r\nENDPOINTS\r\n");
deviceids_count = CheckResult(ibApi_GetDeviceList(api_hdl, groupid,
  ibApi_DEVICETYPE_ENDPOINT, deviceids,DEVICEIDS_MAX));

if (deviceids_count > DEVICEIDS_MAX)
  deviceids_count = DEVICEIDS_MAX;

for (i=0; i<deviceids_count; ++i) {
```

```
    CheckResult(ibApi_Utils_ConvertDeviceIdToText(deviceids[i],
      deviceid_text,sizeof(deviceid_text),MIN_DEVICEID_WORDS));

    sampling_interval = CheckResult(ibApi_GetSamplingInterval(api_hdl,deviceids[i]));

    printf("  %10s (%i ms)\r\n", deviceid_text, sampling_interval);
  }
}

/*****************************************************************************/
int main() {
  /*
   * This handle represents the current API session.
   */
  ibApi_APIHANDLE api_hdl;
  ibApi_VERSION api_version;

  api_version = ibApi_GetApiVersion();
  printf("\r\nInitializing API Version %i.%i.%i\r\n\r\n",
    ibApi_GET_VERSION_MAJOR(api_version),
    ibApi_GET_VERSION_MINOR(api_version),
    ibApi_GET_VERSION_RELEASE(api_version)
  );

  /*
   * ibApi_Open() is called to begin the session.  Your application
   * should ensure that ibApi_Close() is called to release the handle
   * before exiting.
   */
  api_hdl = CheckResult(ibApi_Open(ibApi_EXPECTED_VERSION,"local",""));

  ListDevices(api_hdl);

  CheckResult(ibApi_Close(api_hdl));

  WaitForKey();

  return 0;
}
```

B

Sample Application

This chapter contains information on how to perform the sample application included with the EK-5209-5 Evaluation Kit:

- 'Application Overview' on page B-2
- 'Application Setup & Operation' on page B-4
- 'Changing Temperature Sensor Battery' on page B-7

Application Overview

Millennial Net includes a sample application in the EK-5209-5 Evaluation Kit. This application demonstrates a real-world application of the Millennial Net's endpoint peripheral interface and how it is used for monitoring and data collection purposes across a wireless network.

The EK-5209-5 Evaluation Kit includes a temperature sensor assembly with a 10,000 Ohm Kele sensor (Type 3) that changes resistance as its environmental temperature changes. The sensor can be used to measure temperatures from -35° to +240° degrees Fahrenheit, with precision of 5%.

The sensor's resistance change is measured via voltage change, measured by an endpoint's ADC channel 0, and converted by the TempMonitor application software (included with kit) to temperature. The sensor connection diagram is shown in Figure B-1 and process flow is shown in Figure 2.

Figure B-1. Temperature sensor assembly overview (cover removed)

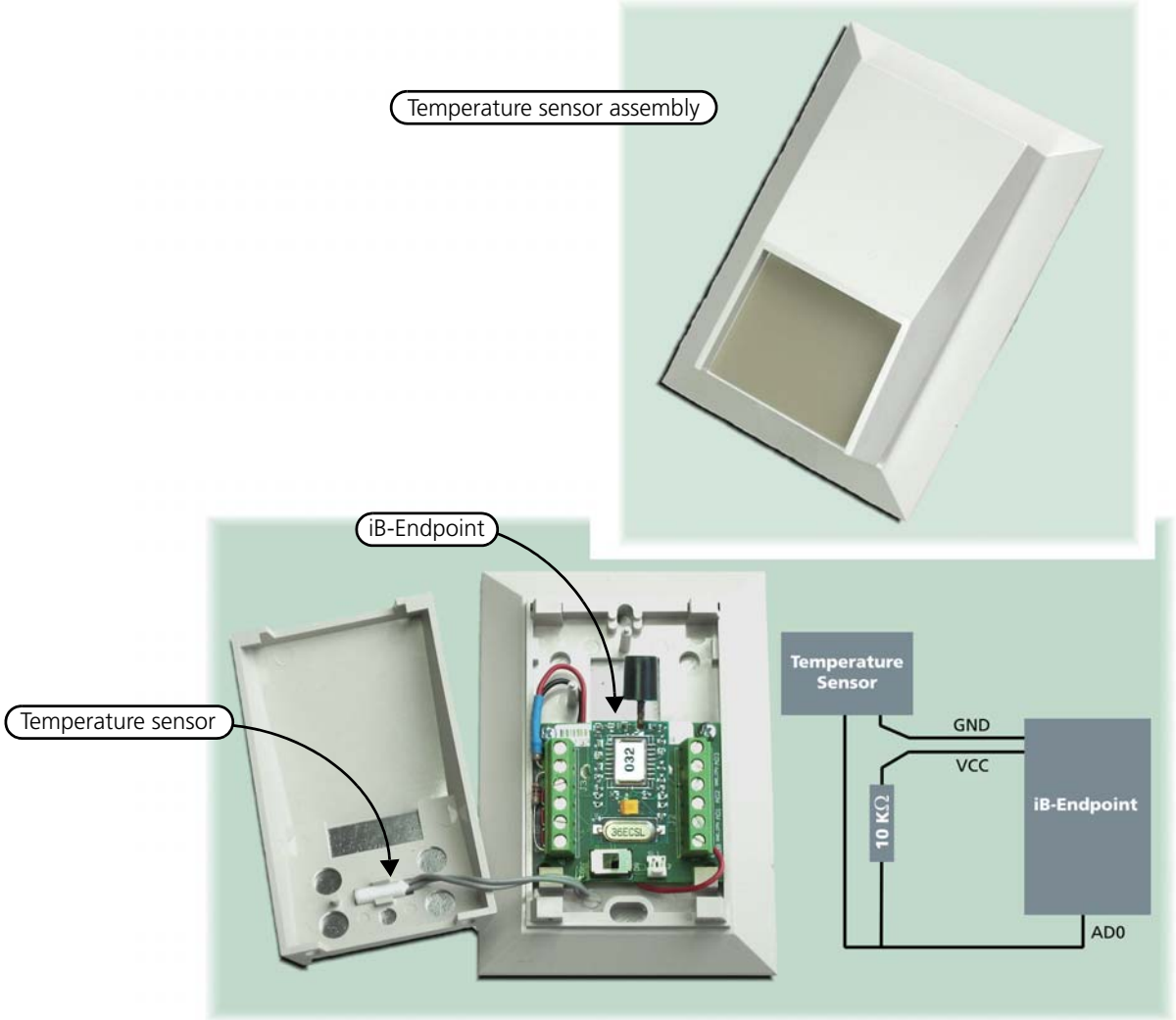
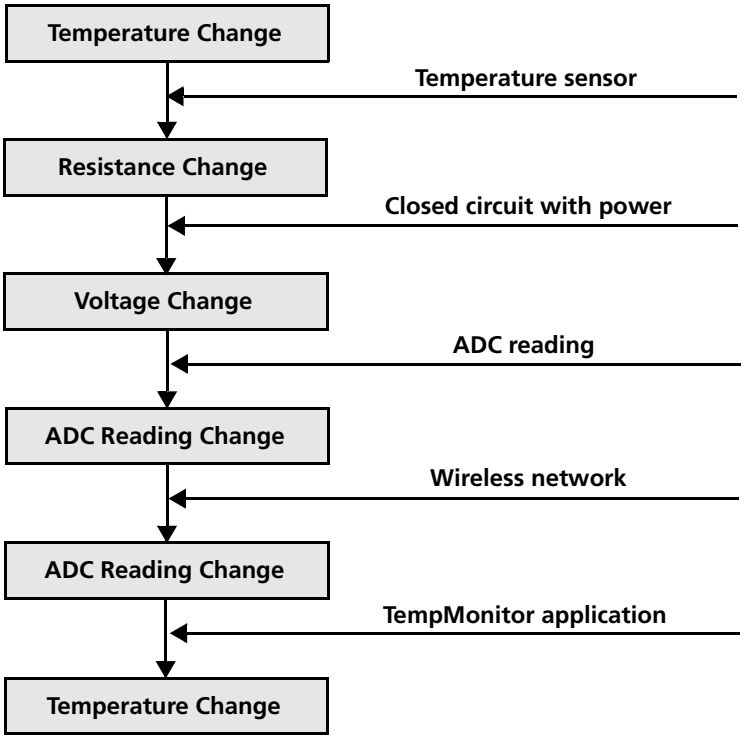


Figure B-2 describes the process flow.

Figure B-2. Process flow



Application Components

In addition to the i-Bean Gateway and iB-5209 Network Monitor software supplied with the EK-5209-5 Evaluation Kit, the following items are included for the sample application:

- **i-Bean Endpoint** (removed from one of the kit terminal boards)
- **Temperature sensor assembly**: This assembly contains a factory installed terminal board and temperature sensor. The terminal board is also equipped with a factory-installed battery for providing power to the i-Bean Endpoint. (i-Bean Endpoints

---

Note: Before using the temperature sensor assembly for this sample application, you will need to install an i-Bean Endpoint (see 'Temperature Sensor Assembly Setup' on page B-4)

- **TempMonitor application software** (automatically loaded on host PC when contents of kit CD are installed)

Application Setup & Operation

The following procedures describe how to set up the iB-5209 hardware and software for creating the evaluation kit's sample application.

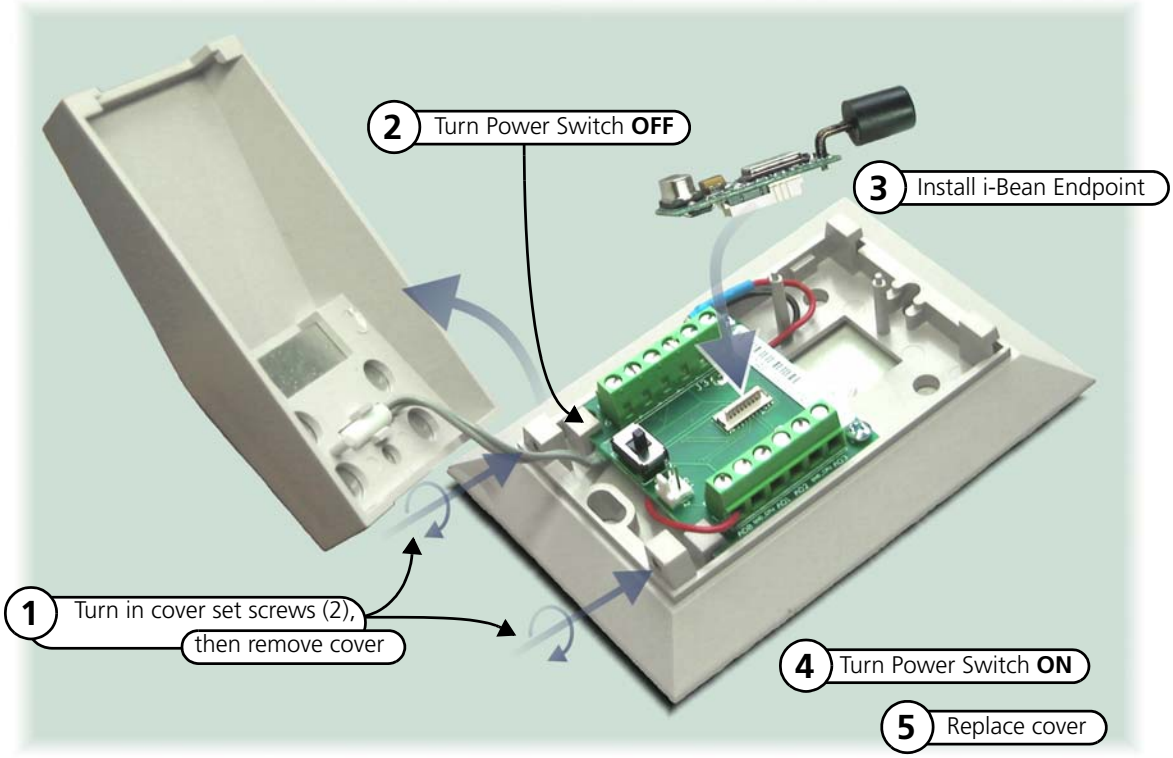
Note: Before this sample application can be executed, you must install the iB-5209 Evaluation Kit's hardware and software according to the instructions found in Chapter 2, "Kit Installation". *The i-Bean Gateway must be installed and powered on, and iB-5209 Network Monitor must be running before proceeding.*

Temperature Sensor Assembly Setup

To install an i-Bean Endpoint in the temperature sensor assembly (see Figure B-3):

1. Use an Allen wrench to turn in the two set screws holding the cover to the base and remove the cover.
2. Turn the Power Switch **OFF**.
3. Remove an i-Bean Endpoint from one of the kit's terminal boards and install it on the terminal board located on the base of the temperature sensor assembly.
4. Turn the Power Switch to **ON**. The sensor is discovered and displayed by iB-5209 Network Monitor.
5. Replace the cover and back out the two cover set screws to secure the cover in place.

Figure B-3. Installing i-Bean Endpoint in Temperature Sensor Assembly

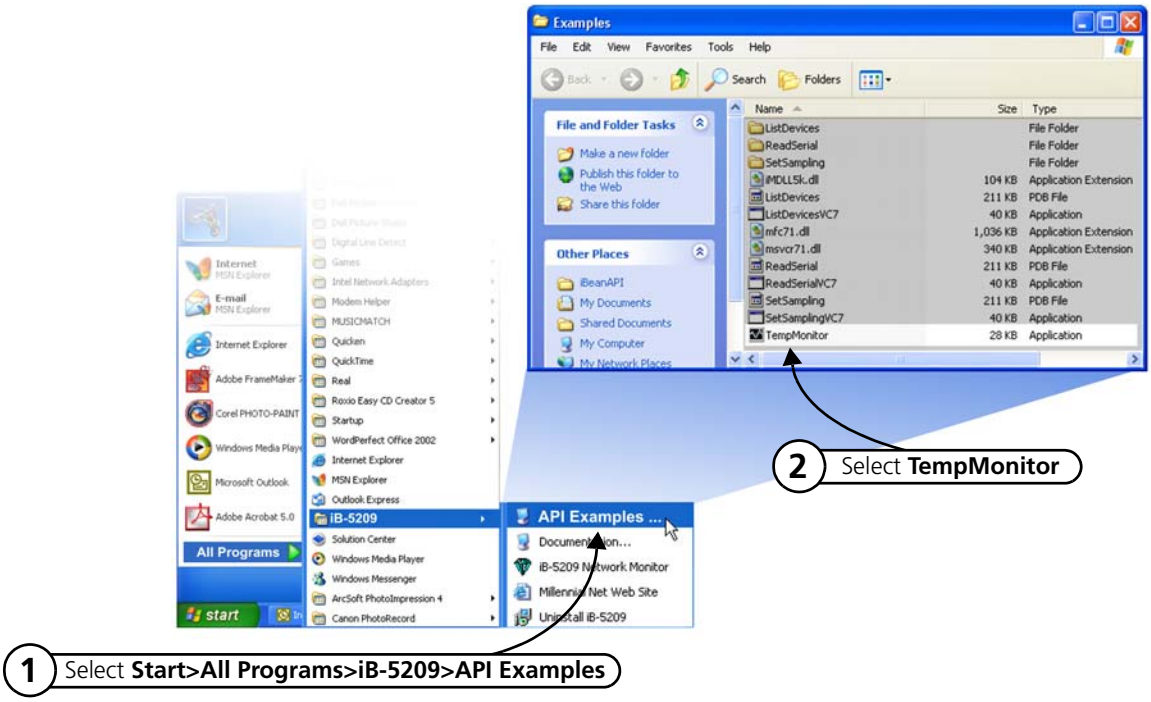


Launching TempMonitor Application

To launch the TempMonitor application (see Figure B-4):

1. From the Windows task bar, select **Star>All Programs>iB-5209>API Examples**. The Examples window opens with a list of options.
2. Select **TempMonitor** from the Examples window. TempMonitor launches.

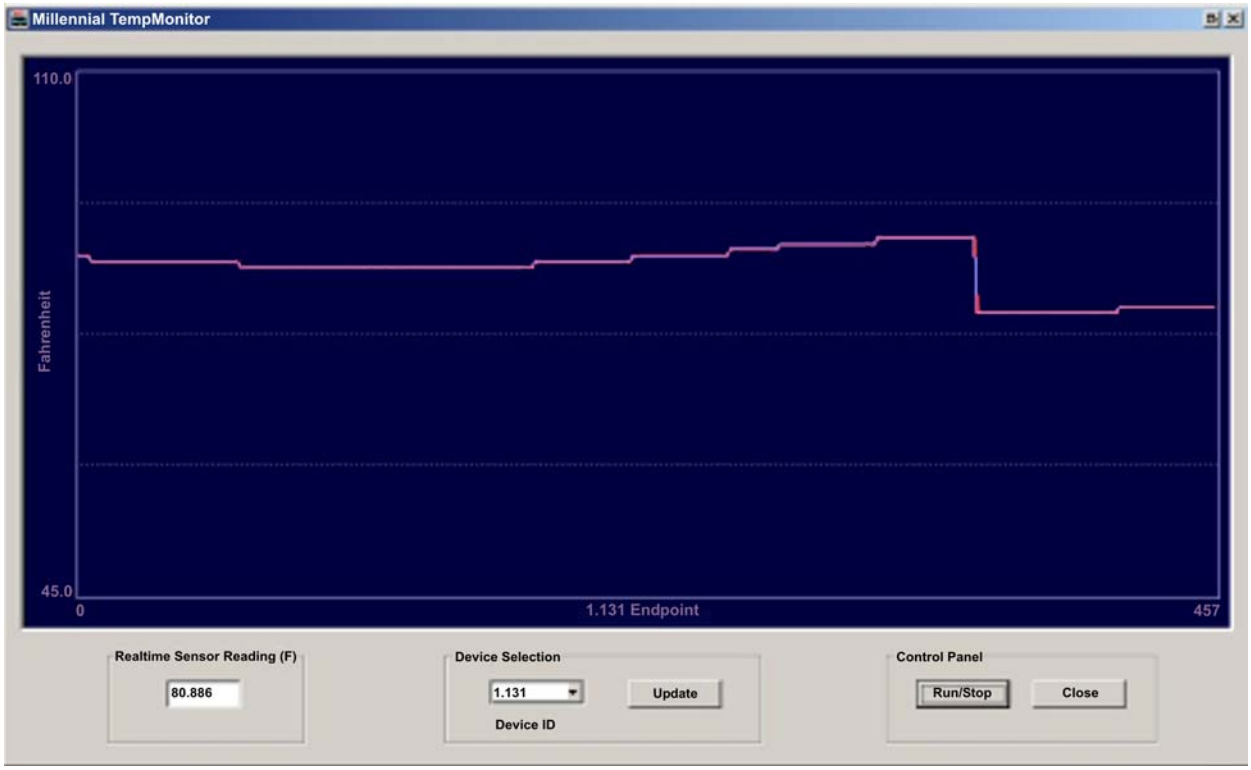
Figure B-4. Launching TempMonitor



TempMonitor Overview

The TempMonitor application displays temperature changes vs. time for one device only. Use the *Device Selection* area to select the endpoint with matching device ID, then click **Update** button to configure the device (enable ADC channel 1). The **Run/Stop** button is used to control the display curve updating; **Run** means continuously update the curve, **Stop** means to freeze the current curve. The precise reading can also be obtains from the *Realtime Sensor Reading(F)* area.

Figure B-5. TempMonitor display



You can change the environment temperature to observe changes to the graph's curve.

Changing Temperature Sensor Battery

The temperature sensor assembly comes with a factory installed lithium battery. The procedure below describes how to replace the battery when needed. Millennial Net recommends using the following replacement battery type:
DigiKey Part Number: **P189-ND** (phone: 1-800-DIGI-KEY).

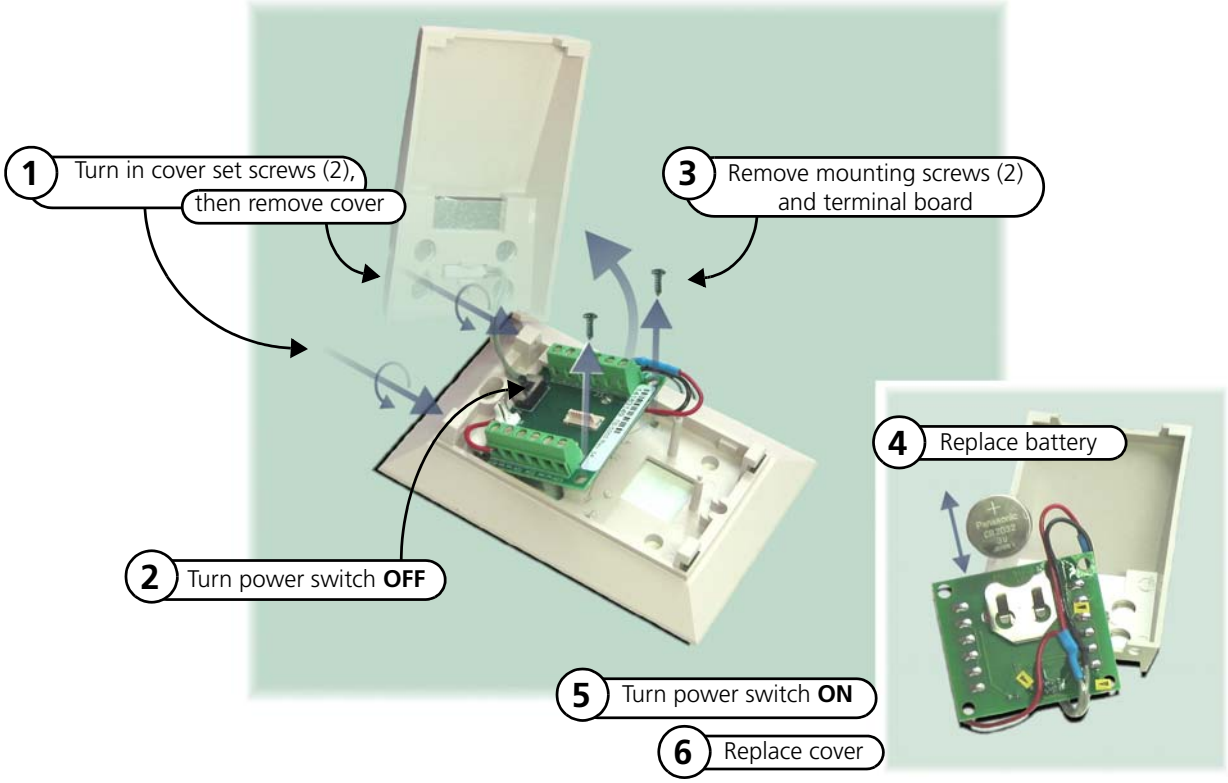
To replace the battery (see Figure B-6):

1. Use an Allen wrench to turn in the two set screws holding the cover to the base and remove the cover.
2. Turn the power switch **OFF**.
3. Remove the screws (2) securing the terminal board to the base and lift the terminal up and off the base.
4. Replace the battery—located on the bottom of the terminal board—with the recommended battery type, observing polarity (**+** side of battery to **+** side of holder).
5. Turn the power switch **ON**.
6. Reinstall the terminal board, replace assembly cover, and back out the two cover set screws to secure the cover in place.

Caution

Do not overtighten the terminal board mounting screws, which may damage the battery holder.

Figure B-6. Changing temperature sensor assembly battery



Index

A

AD Converter
- configuring 3-17

API
- overview B-2

C

com port
- select 3-24

configuring
- ADC 3-17
- digital I/O 3-12
- RS-232 3-18
- RS-485 3-19
- sample interval 3-11
- UART operation 3-15

D

data format
- configure (serial and ADC) 3-23

Digital I/O
- configuring 3-12

digital input setup 3-12

digital output setup 3-13

E

Edit Device window 3-7

ESD warning 1-5

evaluation kit
- contents 1-5
- overview 1-2

event log file
- create 3-21
- view 3-26, 3-28

F

frequency band 1-2

H

hardware installation 2-3

host PC requirements 1-5

I

iB-5209 Network Monitor
- configuring node's operation 3-7
- Counts 3-4
- install software 2-10
- menu bar 3-3
- monitoring features 3-2
- overview 3-2
- sensor node details 3-4

i-Bean
- Endpoint 1-3
- Gateway 1-4
- Router 1-3

i-Bean network overview 1-3

install
- iB-5209 Network Monitor 2-10
- i-Bean Endpoint 2-9
- i-Bean Gateway 2-3
- i-Bean Router 2-6

L

label nodes 3-20

M

major features 1-2

monitor statistics 3-25

multiple capture 3-28

N

Network Monitor
- i-Bean Gateway details 3-4

network topologies 1-4

node status 3-4

P

persistence
- configure 3-22

R

RS-232
- configuring 3-18

RS-485
- configuring 3-19

S

sample application B-2

sample interval
- all nodes 3-11
- single node 3-11

serial communication parameters 3-15

T

temperature sensor assembly
- change battery B-7
- overview B-2
- setup B-4

TempMonitor
- launching B-5

overview B-6
Thread Priority 3-6
transmission distances 1-2

U

UART
configuring 3-15

W

Watch
function 3-9
window 3-9