Millennial Net

# MeshScape™

***Commercial- and Industrial-class Wireless Sensor Networks***

# RK-5424-5 Reference Kit

**for 2.4 GHz MeshScape Systems**

**User's Guide**

Document Number: DOC-0063
Revision: 02
Released: December 2005

Millennial Net, Inc.
2 Fourth Avenue
Burlington, MA 01803-3304 USA
+1 781.222.1030

## CAUTION

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

## Compliance Statements

### FCC Compliance

FCC compliance for Millennial Net's RK-5424-5 Reference Kit (2.4GHz, 5-3-1) consisting of the following models/components:

- EN-5424 end node
- MN-5424 mesh node
- MG-5424 MeshGate gateway

### Compliance Statement (Part 15.19)

The Millennial Net RK-5424-5 Reference 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.

### Note (Part 15.105(b))

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

**Unlicensed Modular Approval (for OEMs)**

The URM-G-2400 and URM-M-2400 comply with the FCC's 47CFR Part 15 rules and regulations as well as Part 15 Unlicensed Modular Approval as outlined in DA 00-1407. Compliance with the Modular Approval rules allows an OEM to integrate the URM-G-2400 and URM-M-2400 into other products without further FCC certification of the intentional radiator, but an OEM must still test their final product to comply with unintentional radiator requirements of 47CFCR Part 15.

Under the Modular Approval rules, an OEM must comply with the following when integrating the URM-G-2400 and URM-M-2400 into an end product:

1. The OEM must ensure that FCC labeling requirements are met. This shall include a clearly visible label on the exterior of the end product with the following nomenclature:

   Contains FCC ID: R8N-URM-G-2400 or R8N-URM-M-2400

2. The OEM must only use the reverse polarity-SMA (RP-SMA) antennas listed below when integrating the device into an end product. These antennas have been tested and approved for use with the URM-G-2400 and URM-M-2400. Integrating the module using any other antenna will require testing to ensure compliance with FCC rules and regulations.

   Centurion ½ wave Antenna   Part Number: WCR2400SMRP

3. The OEM must use the same cable type and of the same length or longer than that defined below. Use of another cable type or of a shorter length will require testing to ensure compliance with FCC rules and regulations.

| | |
|---|---|
| RF Cable type: | RG174 |
| RF Cable Length: | 5.9" +/- 0.13" |
| Millennial Net Cable P/N: | CBL-0018-01 |

**Industry Canada Compliance Statement**

This device has been designed to operate with an antenna having a maximum gain of 2.65 dB. Antenna having a higher gain is strictly prohibited per regulations of Industry Canada. The required antenna impedance is 50 Ohms.

To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (EIRP) is not more than that required for successful communication.

**OEM Integration**

The modules have the same requirements for integration into an OEM product for Industry Canada as it does for FCC. The only difference being the labeling nomenclature required on the exterior of the OEM product. The following must be clearly visible on the exterior of the OEM product:

Contains IC: 5172A-URMG2400 or 5172A-URMM2400

**For countries not covered by FCC Part 15, Industry Canada RSSS-210, or CE**

The RK-5424 -5 Reference Kits are to be used solely by professional engineers for the purpose of evaluating the feasibility of low-power wireless data communications applications. The user's evaluation must be limited to use of an assembled Kit within a laboratory setting which provides for adequate shielding of RF emission which might be caused by operation of the Kit following assembly. In field testing, the assembled device must not be operated in a residential area or any area where radio devices might be subject to harmful electrical interference. Distribution and sale of the Kit is intended solely for use in future development of devices which may be subject to FCC regulation, or other authorities governing radio emission. This Kit may not be resold by users for any purpose. Accordingly, operation of the Kit in the development of future devices is deemed within the discretion of the user and the user shall have all responsibility for any compliance with any authority governing radio emission of such development or use, including without limitation reducing electrical interference to legally acceptable levels. All products developed by user must be approved by the authority governing radio emission prior to marketing or sale of such products and user bears all responsibility for obtaining the approval as needed from any other authority governing radio emission. If user has obtained the Kit for any purpose not identified above, including all conditions of assembly and use, user should return Kit to Millennial Net, Inc. immediately.

## Trademarks

# Contents

# Figures

# Tables

# About This Guide

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

## Audience

This guide is intended for the following qualified service personnel who are responsible for installing, operating, and developing software to interface with the RK-5424-5 MeshScape Wireless Sensor Network Reference Kit:

- System installer
- Hardware technician
- System operator
- System administrator
- Software developer

## Using This Guide

The sections of this guide provide the following information:

| Section | Provides |
|---|---|
| Chapter 1, "Introduction" | General overview of wireless sensor networking and the MeshScape™ system. |
| Chapter 2, "Installing the MeshScape System" | Instructions for installing the components of the RK-5424-5 MeshScape Wireless Sensor Network Reference Kit (MeshGate, Mesh Nodes, End Nodes) and MeshScape Network Monitor (GUI). |
| Chapter 3, "Running MeshScape Network Monitor" | Procedures for using MeshScape Network Monitor software to configure the MeshScape system nodes. Also includes information for attaching external I/O devices to an End Node or Mesh Node. |
| Chapter 4, "Using the MeshScape API" | Information on the MeshScape API functions. |
| Appendix A, "Running the Demo Application" | Procedure for running the sample application provided with the reference kit. |
| Appendix B, "Using MeshScape Programmer" | Instructions for using the MeshScape Programmer application to upgrade the firmware on MeshScape devices, reprogram the group and device IDs, and select the channel on which the devices operate. |
| Glossary | Defines terminology associated with wireless sensor networking and the MeshScape system. |
| 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**:
+1 781.222.1030

**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.

## Additional Resources

To obtain additional resources and information about wireless sensor networking and the development and deployment of MeshScape-based applications, visit the resources page on our Web site at:

www.millennialnet.com/resources

There you will find links to:

- Application notes
- Articles
- Brochures and data sheets
- Case studies
- Industry notes
- Source book
- White papers

# 1

# Introduction

This chapter provides an overview of the MeshScape system and Reference Kit. In this chapter you will find:

# Wireless Sensor Networking Overview

This section provides you with a basic understanding of wireless sensor network concepts and components.

## Defining Wireless Sensors Networks

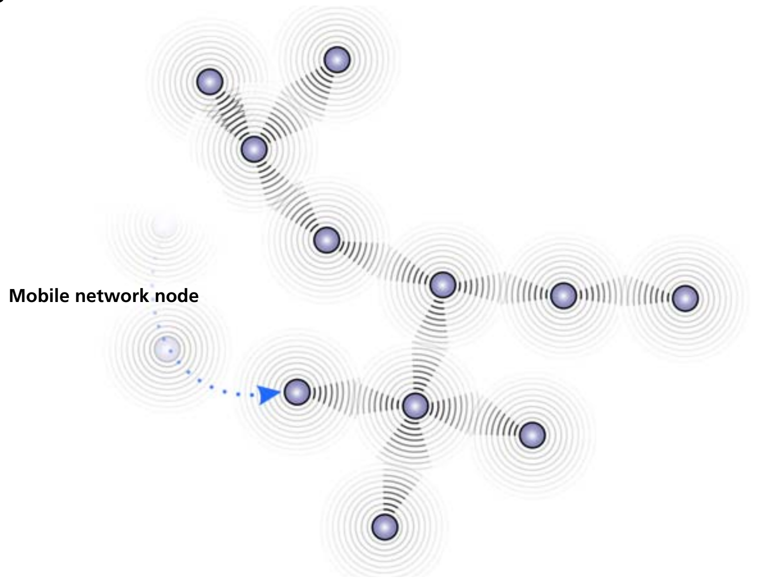
Until recently, networks designed for monitoring and controlling sensors or actuators on a network were limited in application and scope due to a major network design consideration—the cables required to connect the various sensors and actuators to a centralized collection point. In addition to the costs associated with installing and maintaining communication cables (fiber optic or copper), this type of network infrastructure prevents sensor mobility and severely limits the feasible applications of such a network.

Thanks to significant advances in low-power radio and digital circuit design, self-organizing wireless sensor networks are now a reality. Sensors of all types (temperature, motion, occupancy, vibration, etc.) can now be wirelessly enabled and deployed inexpensively and quickly.

Wireless sensor networks fundamentally change the economics of deploying and operating a sensor network, unlocking opportunities to achieve new efficiencies in applications such as production processes, building control, or monitoring. Wireless sensor networks also enable the development of a brand new class of applications and services not previously possible with wired sensor networks.

As illustrated in Figure 1-1, wireless sensor networks form what is called a wireless ad hoc network, which refers to a network's ability to self-organize and self-heal. This means there are no administrative duties associated with establishing and maintaining a wireless sensor network. By comparison, a wired infrastructure network, such as the LAN found in most office environments, requires a significant amount of overhead to install and maintain in terms of cabling and administrative time.

**Figure 1-1. Untethered, mobile ad hoc network nodes**



In an ad hoc network, sensor nodes consisting of a sensor attached to a wireless module can be randomly placed and moved as needed. If the network needs to scale up, additional sensor nodes are easily added. The new sensor nodes and surrounding network will do the work of discovering each other and establishing communication paths through single- and multi-hop paths. All this is made possible through the use of robust, efficient network protocols developed specifically for wireless sensor networks.

## Wireless Sensor Network Components

This section describes the software and hardware that comprise a wireless sensor network.

### System Software

The software required to integrate and operate a wireless sensor network resides as firmware in the system modules and in the application platform as a set of API functions or network monitoring system (NMS).

#### *Module Firmware*

Module firmware is a small, efficient piece of code that incorporates the module into a larger ad hoc network. It “drives” the module's operation as part of the larger ad hoc network.

The firmware is also responsible for packaging the analog and digital sensor data into digital packets and delivering them across the wireless sensor network.Firmware is pre-programmed onto every MeshScape component. However, you have the opportunity to modify certain network parameters for each component. See Appendix B for more information.

#### *API*

An API, or application programming interface, is a set of commonly used functions for streamlining application development. Used by application developers, an API provides hooks to integrate the application platforms with the modules on the wireless sensor network.

API functions are grouped into “libraries.” In wireless sensor networks, there are two different API libraries:

- High-level library: These functions are used to integrate the application with the gateway module.
- Low-level library: These functions are used to integrate the sensor/actuator with the end node module.

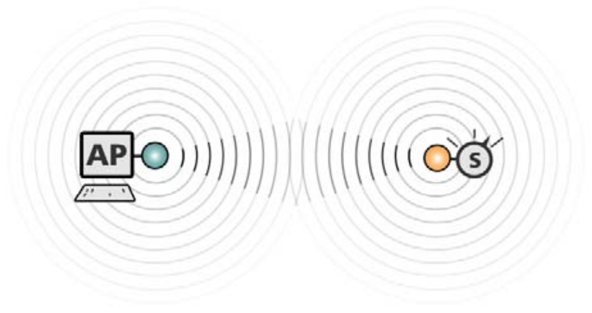
#### *Network Monitoring System*

A network monitoring system (NMS) is software used to interface with a particular wireless sensor network, eliminating the need for any programming. Through the NMS’s graphical user interface (GUI), network operators are able to see the various nodes of their wireless sensor network. Depending on the type of network, control commands can also be issued through the NMS. For example, a pin on a digital interface between an end node and an actuator can be set to high to change the state of the actuator.

## System Modules

The modules of a wireless sensor network enable wireless connectivity within the network, connecting an application platform at one end of the network with one or more sensor or actuator devices at the other end. As shown in Figure 1-2, the gateway and end node modules create a transparent, wireless data path between the application platform and sensor.

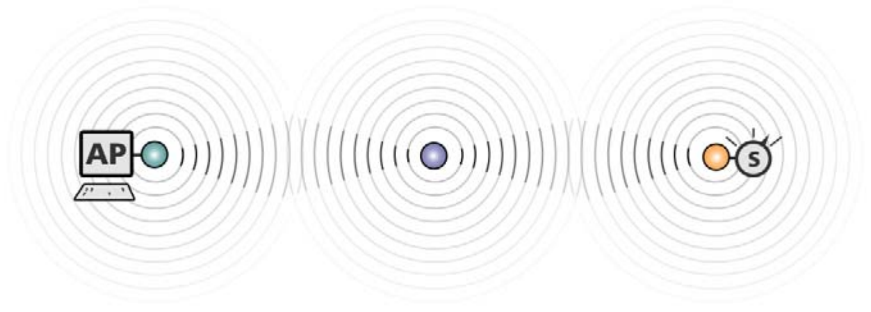
**Figure 1-2. Basic wireless sensor network components**



Exchange of analog or digital information between an application platform and one or more sensor nodes takes place in a wireless fashion. In this example, the data path between the gateway and end node is referred to as a *single-hop* network link.

To extend the range of a network or circumvent an obstacle, a wireless mesh node module can be added between a gateway and an end node as shown in Figure 1-3.

**Figure 1-3. Adding a mesh node module**



This particular example represents a multi-hop data path, in which data packets are handed off from one module to the next before reaching their destination (gateway-to-mesh node-to-end node and vice versa).

More elaborate network layouts are discussed later in Network Topologies," but for now, we'll take a closer look at each of the network components shown in Figure 1-3.

### *Application Platform*

This is the network device (network controller, PC, handheld, etc.) used to monitor and control the actions of the various sensors and actuators that are connected to the wireless sensor network. The application platform is capable of making decisions based on the information it gathers from the network. Typically, the wireless sensor network will come with an API and/or a GUI used to interface with the wireless modules.

### *Gateway*

The gateway is the interface between the application platform and the wireless nodes on the network. The gateway can be a discrete module, or it can be integrated onto a Flash card form factor for use in, for example, a handheld device. All information received from the various network nodes is aggregated by the gateway and forwarded on to the application platform. In the reverse direction, when a command is issued by the application program to a network node, the gateway relays the information to the wireless sensor network. The gateway can also perform protocol conversion to enable the wireless network to work with other industry-standard network protocols.

### *Mesh Node Module*

Considered full-function devices (FFD), mesh node modules (sometimes called routers) are used to extend network coverage area, route around obstacles, and provide back-up routes in case of network congestion or device failure. In some cases, mesh nodes may also be connected via analog and digital interfaces to sensors and actuators, providing the same I/O functionality of an end node module. Mesh nodes can be battery powered or line powered.

### *End Node Module*

Considered reduced-function devices (RFD), end nodes (sometimes called endpoints) provide the physical interface between the wireless sensor network and the sensor or actuator to which it is wired. End nodes will usually have one or more I/O connections for connecting to and communicating with analog or digital sensor or actuator devices. End nodes are typically battery powered.

### *Sensor/Actuator*

These are the devices you ultimately wish to monitor and/or control. An example is a sensor monitoring the pressure in an oil pipeline.

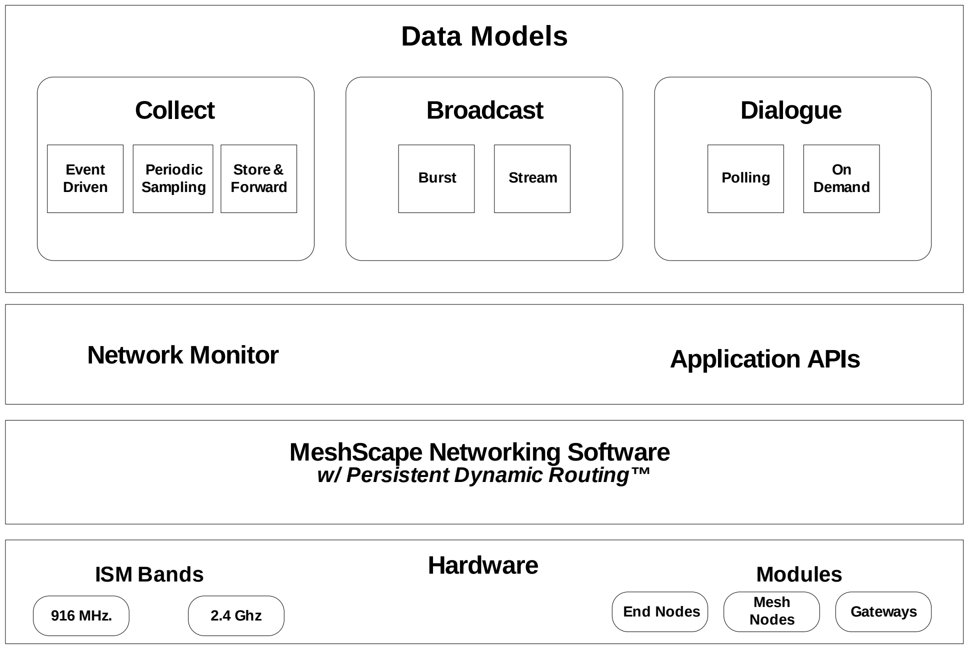
# MeshScape System Overview

In order to realize benefits wireless sensor networking promises, the technology must be able to address several critical requirements: reliability of data transmission, responsiveness to adapt to dynamic environments, power efficiency, and scalability. The MeshScape™ wireless sensor networking system from Millennial Net delivers on all of these requirements. The MeshScape ready-to-embed hardware modules and assemblies support fast and cost-effective application development.

## Core Elements of MeshScape System

The core elements of the MeshScape wireless sensor networking system are depicted in Figure 1-4 below.

**Figure 1-4. MeshScape system core elements**



### MeshScape Networking Software

The ultra-efficient, highly scalable, self-organizing networking software is based on Persistent Dynamic Routing™ techniques. The networking software is delivered on the hardware modules described in this section. For volume applications, the MeshScape system software can also be licensed and integrated directly onto your sensor assembly.

Millennial Net has developed and optimized its protocol to address the unique characteristics and challenges associated with wireless sensor networking. The end result is a networking system and associated protocol that is highly scalable, ultra-efficient, and extremely responsive and resilient in dynamic environments. The MeshScape protocol for wireless sensor networks provides the industry's longest battery life at sensor nodes while delivering data over fault-tolerant links with end-to-end redundancy. The Millennial Net protocol is based on a set of techniques including, Persistent Dynamic Routing for reliable and scalable wireless sensor networks. When forming an ad hoc sensor network, Persistent Dynamic Routing requires minimal overhead for requesting and establishing connectivity without relying on the bandwidth-consuming flooding technique.

### MeshScape Network Monitor and Application APIs

The MeshScape system delivers the tools to view and control network dynamics. The MeshScape Network Monitor provides functions for monitoring and managing the network. Application APIs streamline development by providing input/output functions for sensor and application integration.

### Hardware

The MeshScape system includes field-proven, "integratable" modules for fastest time to market. These ready-to-integrate end nodes, mesh nodes, and gateways support numerous application requirements and support various ISM bands for license-free operation around the world.

## Data Models

The MeshScape system provides built-in support for data movement profiles to speed development including:

- data collection models
- bi-directional dialogue models
- broadcast models

These data models optimize the network for an application's specific data requirements and support a variety of classes for collection and bi-directional dialogue data models.

### Data Collection Models

Data collection models describe monitoring applications where the data flows primarily from the sensor node to the gateway. The MeshScape system supports the data collection models described in this section.

#### *Periodic Sampling*

For applications where certain conditions or processes need to be monitored constantly, such as the temperature in a conditioned space or pressure in a process pipeline, sensor data is acquired from a number of remote sensor nodes and forwarded to the gateway or data collection center on a periodic basis.

The sampling period mainly depends on how fast the condition or process varies and what intrinsic characteristics need to be captured. In many cases, the dynamics of the condition or process to be monitored can slow down or speed up from time to time. Therefore, if the sensor node can adapt its sampling rate to the changing dynamics of the condition or process, over-sampling can be minimized and power efficiency of the overall network system can be further improved.

Another critical design issue associated with periodic sampling applications is the phase relation among multiple sensor nodes. If two sensor nodes operate with identical or similar sampling rates, collisions between packets from the two nodes is likely to happen repeatedly. It is essential that sensor nodes can detect this repeated collision and introduce a phase shift between the two transmission sequences in order to avoid further collisions resulting in optimal network operation and minimized power usage.

#### *Event Driven*

There are many cases that require monitoring one or more crucial variables immediately following a specific event or condition. Common examples include fire alarms, door and window sensors, or instruments that are user activated. To support event-driven operations with adequate power efficiency and speed of response, the sensor node must be designed such that its power consumption is minimal in the absence of any triggering event, and the wake-up time is relatively short when the specific event or condition occurs. Many applications require a combination of event driven data collection and periodic sampling.

#### *Store and Forward*

In many applications, data can be captured and stored or even processed by a sensor node before it is transmitted to the gateway or base station. Instead of immediately transmitting every data unit as it is acquired, aggregating and processing data by remote sensor nodes can potentially improve overall network performance in both power consumption and bandwidth efficiency. One example of a store-and-forward application is cold-chain management where the temperature in a freight container carrying produce or pharmaceuticals, for instance, is captured and stored; when the shipment is received, the temperature readings from the trip are downloaded and viewed to ensure that the temperature and humidity stayed within the desired range.

### Bi-Directional Dialogue Data Models

Bi-directional dialogue data models are characterized by a need for two-way communication between the sensor/actuator nodes and gateway/application. The MeshScape system supports the bi-directional dialogue data models described in this section.

#### *Polling*

Controller-based applications, such as those found in building automation systems, use a polling data model. In this model, there is an initial device discovery process that associates a device ID with each physical device in the network. The controller then polls each device on the network successively, typically by sending a serial query message and waiting for a response to that message. For example, an energy management application would use a polling data model to enable the application controllers to poll thermostats, variable air volume sensors, and other devices for temperature and other readings.

#### *On-Demand*

The on-demand data model supports highly mobile nodes in the network where a gateway device enters the network, automatically binds to that network and gathers data, then leaves the network. With this model, one mobile gateway can bind to multiple networks and multiple mobile gateways can bind to a given network. An example of an application using the on-demand data model is a medical monitoring application where patients in a hospital wear sensors to monitor vital signs and doctors access that data via a PDA that is a mobile gateway. A doctor enters a room and the mobile PDA automatically binds with the network associated with that patient and downloads vital sensor data. When the doctor enters a second patient's room, the PDA automatically binds with that network and downloads the second patient's data.

### Broadcast Data Models

Broadcast data models are characterized by a need for one-to-many communication between the gateway/application and sensor/actuator nodes. The MeshScape system supports the broadcast data models described in this section.

#### *Burst*

The burst data model is characterized by an uneven pattern of data transmission from the gateway/application to all sensor/actuator nodes on the wireless sensor network. The burst data model has been used with industrial lighting applications.

#### *Stream*

In the stream data model, the gateway/application sends data in a continuous stream to all sensor/actuator nodes on the wireless sensor network. The transport service guarantees that all data is delivered to the other end in the same order as sent and without duplicates. The stream data model is used when performing network upgrades.

## Low-Power Configuration

Many sensors are dispersed over a wide area and must rely on batteries or solar cells for their power source. Consider the example of sensors taking measurements on a gas pad, it would be prohibitively expensive to network these sensors using cables, so a wireless sensor network is the perfect solution. However, to be useful in such an environment, the wireless sensor network must posses the following characteristics:

- Power: Low power consumption—sensor and node must be able to operate 10+ years on a singe battery
- Scalability: End nodes must be able to scale as sensor node counts increase.
- Data Rate: Application/gateway must support a configurable sample rate.
- Range: End nodes must be able to communicate over distances of 60 to 80 feet (18 to 24 meters) and Mesh nodes must be able to communicate at distances up to 100 feet (30 meters).
- Integration: The end nodes must be integrated with the sensor.

The MeshScape system possesses all of theses characteristics and uses configurable sleep and duty cycle intervals to minimize power consumption.

# The MeshScape RK-5424-5 Reference Kit

Millennial Net's RK-5424-5 Reference 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 RK-5424-5 Reference Kit hardware includes:

- one MeshGate Gateway
- three mesh nodes
- five end nodes
- connecting cables

Reference kit software includes:

- MeshScape Network Monitor - the MeshScape system network monitoring tool and graphical user interface (GUI)
- MeshScape Programmer application - enables you to upgrade the firmware on MeshGate gateways, mesh nodes, and end nodes, and modify the group and device IDs of deployed mesh nodes and end nodes (see Appendix B, “Using MeshScape Programmer”)
- Application Program Interface (API) library - A complete API library is provided to streamline development using *MS Visual C++.NET* on a PC. For applications where the MeshGate connects to a third-party controller, Millennial Net also provides libraries (pre-compiled Windows API library and Linux API library source), as well as source code examples.

The kit also includes:

- temperature sensor assembly - enables you to run a sample application (see Appendix A, “Running the Demo Application”).

Documentation for the reference kit includes:

- this user’s guide - which describes how to set up the MeshScape network, including connections to the host computer, power supplies, sensors, and other devices
- MeshScape Product Family Sheet
- technical specifications for MeshGate gateway, mesh node, and end node

For complete details on the contents of the reference kit, refer to ’Reference Kit Contents’ on page 1-11.

MeshScape Network Monitor runs on MS Windows XP and allows you to set network and device operating parameters, and monitor the status of the MeshScape components and their inputs/outputs. The API software runs on Windows and Linux systems and can be easily incorporated into user application programs written in C++.

## Major Features

Major features of the MeshScape RK-5424-5 Reference Kit include the following:

- frequency band: 2.4 GHz
- bi-directional/multiple-access communication
- MeshScape Network Monitor graphical user interface (GUI) for configuring the MeshScape system and evaluating its performance
- Application Programming Interface (API)
- end node and mesh node-specific features include:
  - configurable sampling interval
  - digital I/O - 4 channels
  - ADC input - 4 channels
  - UART input/output

## Reference Kit Contents

The MeshScape RK-5424-5 Reference Kit contains the following components:

- (4) EN-5424 end nodes; each end node is mounted to a terminal board equipped with a battery.
- (1) EN-5424 end node mounted to a terminal board equipped with a battery and a Kele temperature sensor for use with the supplied sample MeshScape application.
- (3) MN-5424 Mesh Nodes (enclosed) with AC power adapters.
- (1) MG-5424 MeshGate gateway (enclosed) with an AC power adapter.
- (4) antennas; one 1/2-wave antenna for each Mesh Node and one for the MeshGate.
- (1) RS-232 serial cable for connecting the MeshGate serial port to the host PC. This is a DB-9, male-to-female, straight-through cable.
- (1) RS-232 serial cable for connecting the MeshGate console port to the host PC. This is a DB-9-to-mini-connector cable.
- (1) MeshGate programming cable
- (1) MeshGate-to-end node programming adapter
- (4) International power outlet adapter kits for supplied power adapters
- (1) CD-ROM containing support documentation and application software, including the MeshScape Network Monitor program, MeshScape Programmer application, 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 reference kit requires a personal computer (PC) to run the supplied application software. The host PC must have the following minimal configuration:

- Microsoft Windows XP
- Processor: 1.0 GHz
- 512 MB RAM
- RS-232 serial port
- CD-ROM drive for loading software
- Display with SVGA (800 x 600) resolution
- 10 MB free disk space

Although the above platform is required to run MeshScape Network Monitor and other supplied applications, the supplied API library files are supported on both Windows and Linux platforms.

*Microsoft Visual C++ .NET* is recommended for development purposes on Windows platforms.

2

# Installing the MeshScape System

This chapter provides the following MeshScape system installation information:

## Installing the MeshScape Wireless Sensor Network

This section of the user's guide describes how to install the reference kit's hardware and software components. Installation should be performed in the following order:

1. MeshGate (see 'MeshGate Setup (MG-5424)' on page 2-3)
2. Mesh Nodes (see 'Mesh Node Setup (MN-5424)' on page 2-8)
3. End Nodes (see 'End Node Setup (EN-5424)' on page 2-12)
4. MeshScape Network Monitor (see 'Installing Contents of Millennial Net's RK-5424 CD-ROM' on page 2-15)

Once the hardware is set up and the MeshScape Network Monitor software installed, launch MeshScape Network Monitor to verify that all hardware is detected and displayed.

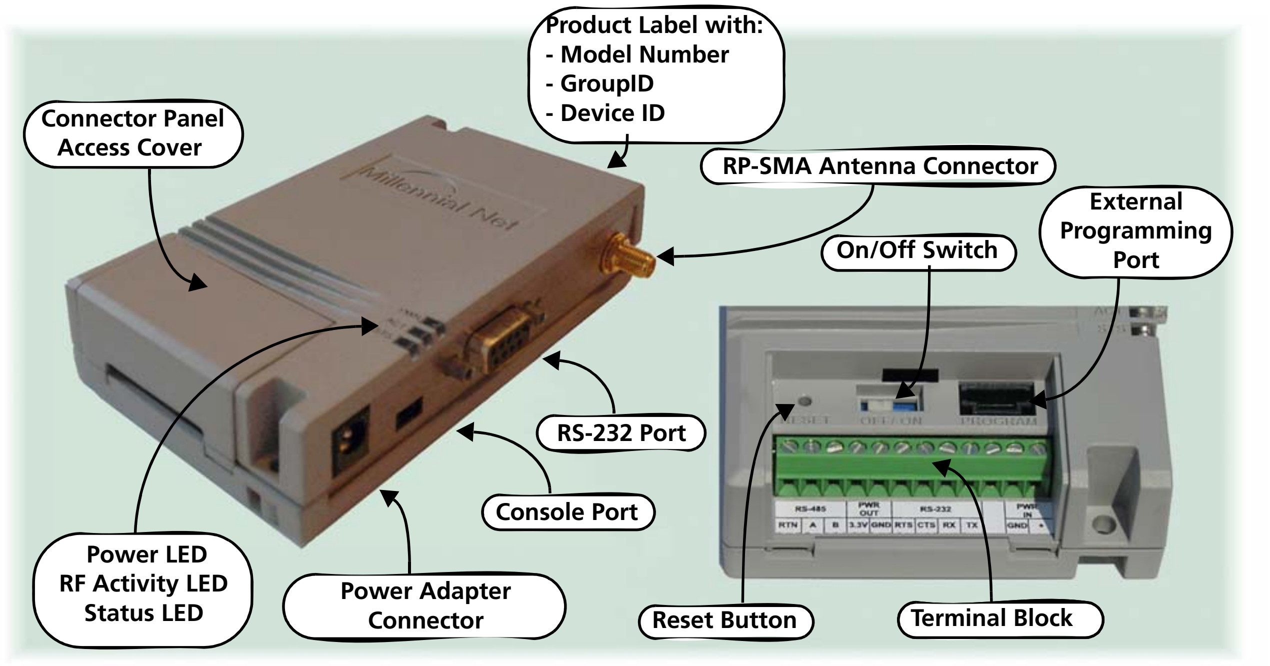
# Installing the Hardware

The following procedures describe in order, how to install the various hardware components of the reference kit. When initially setting up the hardware, it is recommended that the MeshGate, Mesh Nodes, and End Nodes be placed close to the host PC. This will make verifying proper network installation and operation easier when first establishing a session with MeshScape Network Monitor. The devices can then be moved away from the host PC as needed.

## MeshGate Setup (MG-5424)

The MeshGate, model number MG-5424 (label with model number on bottom), is shipped enclosed in a case that provides access to the antenna connector, RS-232 data port, console port, and power connectors as shown in Figure 2-5. Additionally, a lift-off connector panel access cover on the case provides access to a 12-pin terminal block connector, a reset button, an on/off switch, and a 6-pin external programming port.

**Figure 2-5. MeshGate components**



The pin-out for the MeshGate terminal block is as follows:

| RS-485 | | | PWR OUT | | RS-232 | | | | | PWR IN | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RTN | A | B | 3.3V | GND | RTS | CTS | RX | TX | | GND | + |

The function of each MeshGate terminal block pin is described as follows:

**Table 2-1. MeshGate terminal block pin assignments**

| Pin | Label | Input/Output | Function |
|---|---|---|---|
| 1 | RTN | Reference | Reference connection for RS-485 |
| 2 | A | I/O | RS-485 signal + |
| 3 | B | I/O | RS-485 signal - |
| 4 | 3.3V | Output Power | 3.3V output power |
| 5 | GND | Power | Digital ground |
| 6 | RTS | Input | RS-232 Request to Send |
| 7 | CTS | Output | RS-232 Clear to Send |
| 8 | RX | Output | RS-232 Receive Data |
| 9 | TX | Input | RS-232 Transmit Data |
| 10 | N/A | N/A | Not used |
| 11 | GND | Power | Digital Ground |
| 12 | + | Power | Input power (4.5V to 30V) |

During the setup procedure, refer to Figure 2-5 for location of the various MeshGate components.

To set up the MeshGate:

1. Attach one of the four included 1/2-wave antennas to the RP-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 MeshGate's RS-232 Port and the host PC's serial port.
3. Plug the supplied AC adapter into the MeshGate power connector and then into a 110/220 VAC power source.
4. Remove the connector panel access cover and slide the on/off switch to the ON position.
5. Replace the connector panel access cover.

   The MeshGate is ready to interface with the host PC and surrounding network nodes (Mesh Nodes and End Nodes). For information on the behavior of the status LEDs, see Table 2-2.

## Mounting options

There are three mounting options for the MeshGate:

- desktop
- wall
- DIN rail

*Mounting the MeshGate on a Desktop*

1. Choose a level, stable surface on which to rest the MeshGate.
2. Install one of the four supplied self-adhesive rubber feet in the round depression located in each corner on the bottom of the MeshGate chassis.

*Mounting the MeshGate on a Wall*

When mounting the MeshGate to a wall, we recommend that you secure the MeshGate in place using two #6 screws and screw anchors (*not supplied*) of the appropriate type for the mounting surface.

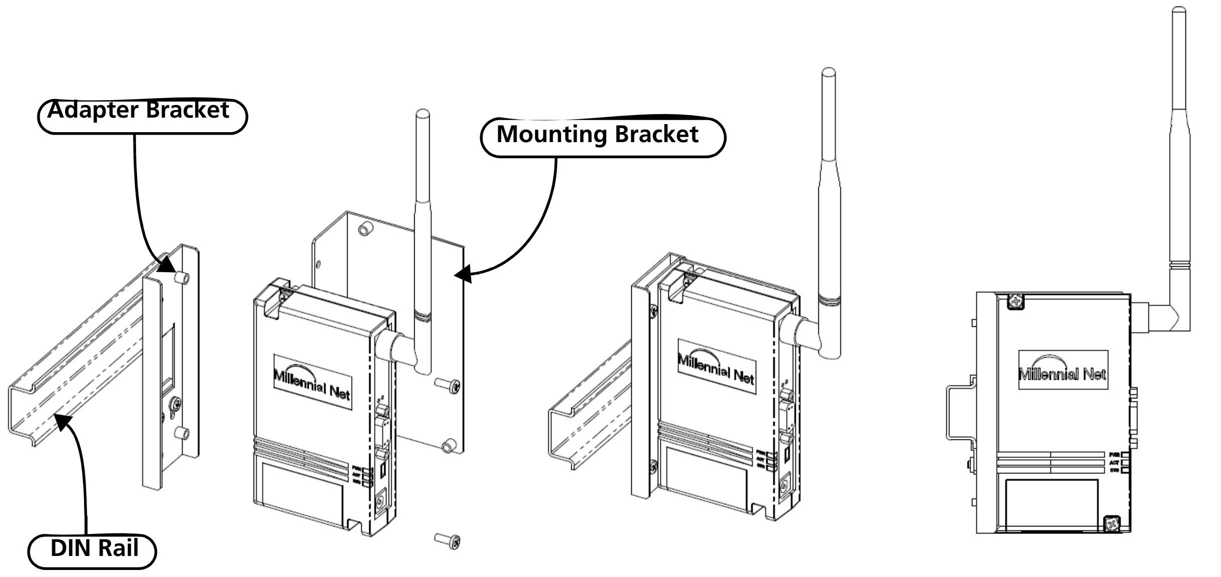
1. Place the MeshGate against the wall in the desired mounting location.
2. Mark the location of the two chassis screw holes on the wall.
3. Drill two screw holes in to the wall at the marked locations.
4. Mount the MeshGate to the wall using two #6 screws (not supplied).

*Mounting the MeshGate to a DIN Rail*

Millennial Net offers an optional DIN rail mounting kit (MG-DIN) to enable you to mount the MeshGate to a standard DIN rail easily and quickly.

To mount the MeshGate to a DIN rail using the supplied DIN rail mounting bracket and hardware, refer to Figure 2-6 and complete the following steps:

**Figure 2-6. Mounting the MeshGate to a DIN rail**



1. Using two of the supplied screws, secure the MeshGate chassis to the mounting bracket.
2. Mount the adapter bracket onto the DIN rail. Slide the adapter bracket's clamp up and then tighten its two screws to secure the adapter bracket in place on the DIN rail.
3. Using two of the supplied screws, secure the mounting bracket to the adapter bracket.

## MeshGate status LED operation

Table 2-2 describes how the status LEDs on the MeshGate behave.

**Table 2-2. MeshGate status LEDs**

| LED | Led State | Status |
|---|---|---|
| PWR | On | Connection with host device detected. |
| | Blinking | No host device detected or MeshScape Network Monitor not running. |
| | Off | Power has been removed. |
| ACT | Flashing | Gateway detects RF activity. The Activity LED 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. |
| | Off | No RF activity detected. |
| STS | *(Reserved for future use.)* | |

## MeshGate default settings

Table 2-3 lists the default settings for the MeshGate gateway.

**Table 2-3. MeshGate default Settings**

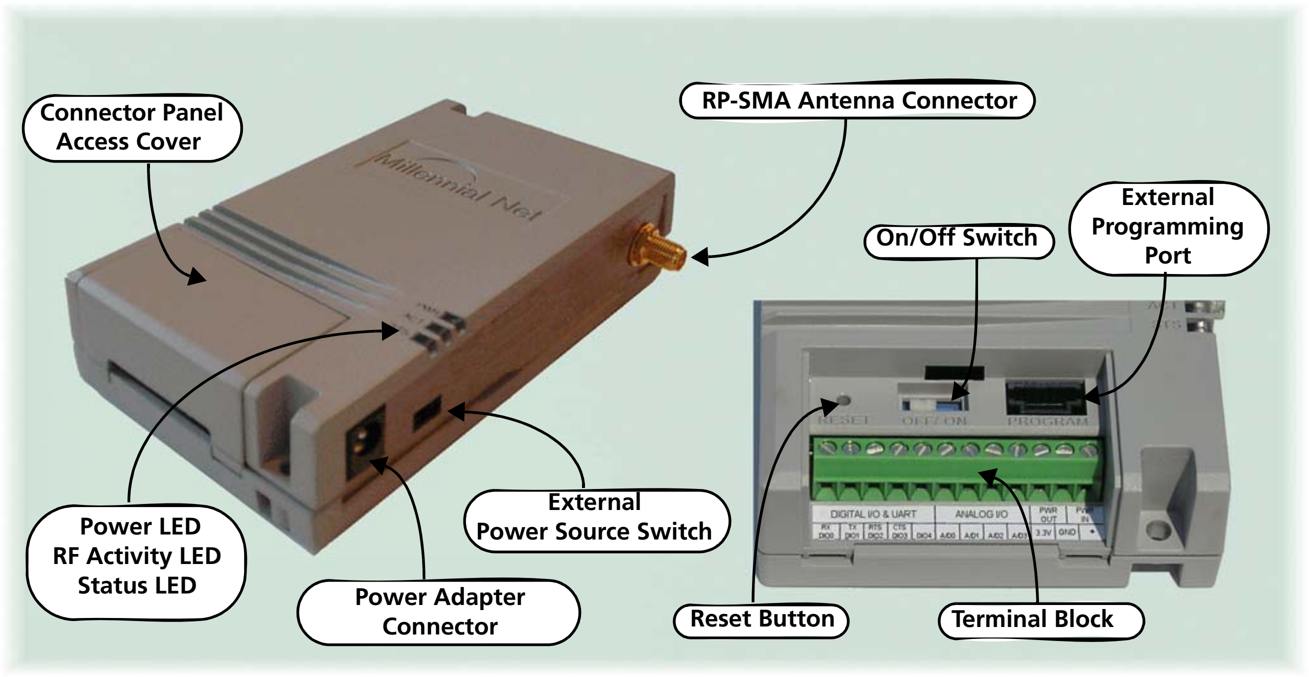
| **Variable** *Description* | **Default Value** | **Persisted?*** |
|---|---|---|
| RS-232 Data Port Configuration | RS232<br>115,200 baud<br>No parity<br>No hardware flow control | Yes |
| Console Port | RS232<br>115,200 baud<br>No parity<br>No hardware flow control | Yes |

* Persisted indicates value is retained after power cycle.

## Mesh Node Setup (MN-5424)

The Mesh Nodes, model number MN-5424 (label with model number on bottom), are shipped enclosed in cases that provide: RP-SMA antenna connector, three-position external power source switch, and a connector for the supplied power adapter as shown in Figure 2-7. Additionally, a lift-off connector panel access cover on the case provides access to a 12-pin terminal block connector, a reset button, an on/off switch, and a six-pin external programming port.

**Figure 2-7. Mesh node components**



The pin-out for the Mesh Node terminal block is as follows:

| DIGITAL I/O & UART | | | | | ANALOG I/O | | | | PWR OUT | | PWR IN |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RX | TX | RTS | CTS | | | | | | | | |
| DIO0 | DIO1 | DIO2 | DIO3 | DIO4 | A/D0 | A/D1 | A/D2 | A/D3 | 3.3V | GND | + |

The function of each Mesh Node terminal block pin is described as follows:

**Table 2-4. Mesh Node terminal block pin assignments**

| Pin | Label | Input/Output | Function |
|---|---|---|---|
| 1 | DIO0/RxD | I/O or Output | Digital Input/Output 0 or UART RX |
| 2 | DIO1/TxD | I/O or Input | Digital Input/Output 1 or UART TX |
| 3 | DIO2/RTS | I/O or Input | Digital Input/Output 2or UART RTS (in) |
| 4 | DIO3/CTS | I/O or Output | Digital Input/Output 3 or UART CTS (out) |

**Table 2-4. Mesh Node terminal block pin assignments**

| Pin | Label | Input/Output | Function |
|---|---|---|---|
| 5 | DIO4 | I/O | Not used. |
| 6 | AD0 | Input | Analog Input 0 |
| 7 | AD1 | Input | Analog Input 1 |
| 8 | AD2 | Input | Analog Input 2 |
| 9 | AD3 | Input | Analog Input 3 |
| 10 | 3.3V | Output Power | 3.3V output power |
| 11 | GND | Power | Digital ground |
| 12 | + | Power | Input power (4.5V to 30V) |

### Installing the antenna and applying power

To install a Mesh Node:

1. Attach one of the four supplied 1/2-wave antennas to the RP-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. Ensure that the Mesh Node External Power Source Switch is in the **DC Ext** position.
3. Plug the supplied AC adapter into the MeshGate power connector and then into a 110/220 VAC power source.
4. Remove the connector panel access cover and slide the on/off switch to the ON position.

   Replace the connector panel access cover.

   The Mesh Node is ready to operate as a router device (see note below). For information on the behavior of the status LEDs, see Table 2-5.
5. Repeat Steps 1 to 4 for each Mesh Node in the kit.

### Mounting options

You may operate the mesh node while it is resting on a desktop or mounted to a wall. When mounting the mesh node to a wall, we recommend that you secure the mesh node in place using two #6 screws and screw anchors (*not supplied*) of the appropriate type for the mounting surface.

### Mesh node status LED operation

Table 2-5 describes how the status LEDs on the Mesh Node behave.

**Table 2-5. Mesh node status LEDs**

| LED | Led State | Status |
|---|---|---|
| PWR | On | Power on. |
| | Off | No power. |
| RF Activity | Flashing | Mesh Node detects RF activity. The RF Activity LED 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. |
| | Off | No RF activity detected. |
| STS | On | The Mesh Node has established two northbound pathways to the MeshGate. |
| | Blinking | The Mesh Node has established a single northbound pathway to the MeshGate. |
| | Off | The Mesh Node is not on the MeshScape network. |

### Mesh node default settings

Table 2-6 lists the default settings for the MeshScape mesh node.

**Table 2-6. MeshScape mesh node default settings**

| Variable<br>*Description* | Default Value | Persisted?* |
|---|---|---|
| Sampling Interval<br>*Defines the time interval between successive sensor samples sent from mesh node to MeshGate* | 300 Seconds | No |
| A/D Converter Setup<br>*Defines state of A/D converter sensor inputs* | Channels 0-3 disabled | Yes |
| DIO Setup<br>*Defines state of digital I/O lines* | DIO<0:3> configured as inputs | Yes |
| Serial Data configuration<br>*Defines configuration of serial data sensor input, options are:*<br>***Off** - DIO<0:3> configured as I/O lines*<br>***Digital UART** - DIO<0:3> configured as digital UART signals* | **Off** - DIO<0:3> defined as I/O lines | Yes |

**Table 2-6. MeshScape mesh node default settings (continued)**

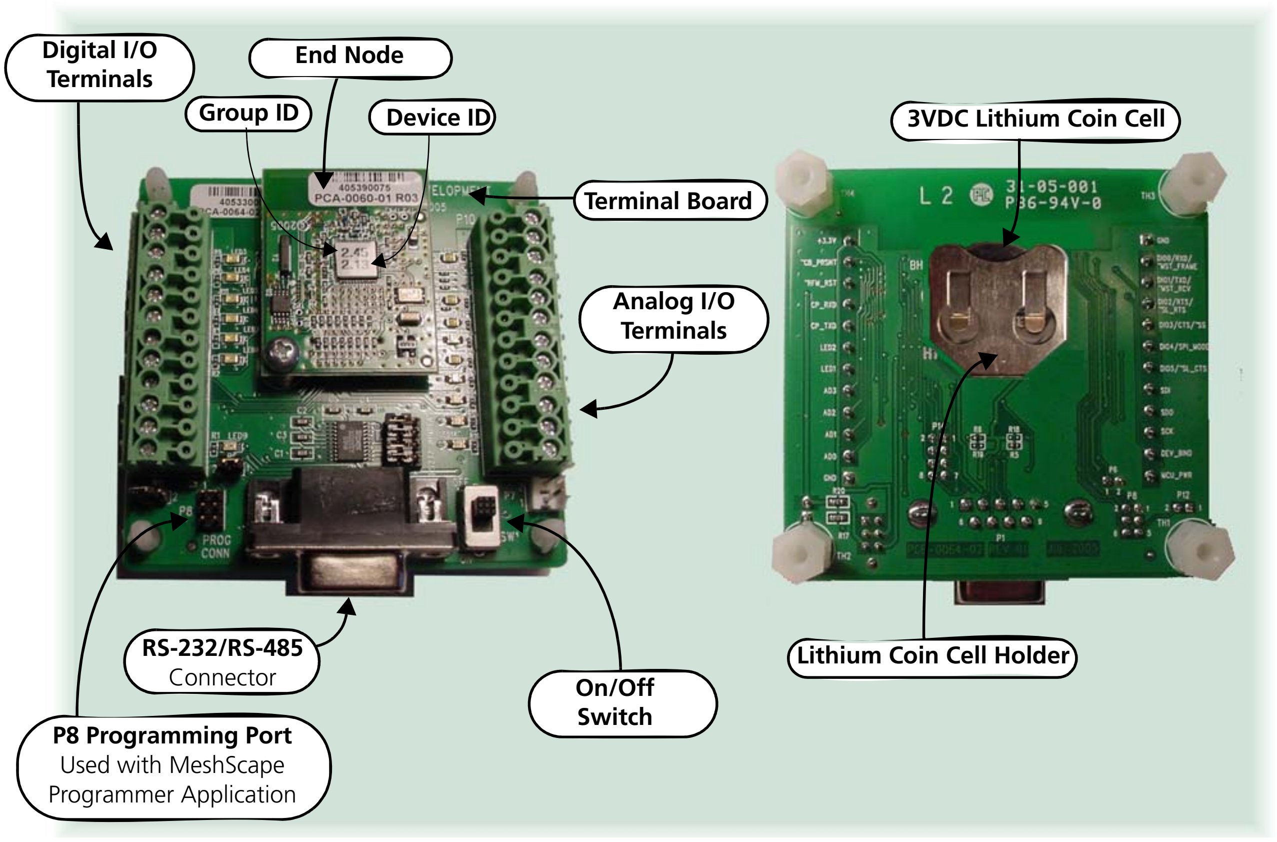
| **Variable** <br> ***Description*** | **Default Value** | **Persisted?*** |
|---|---|---|
| Label <br> *Text label used by Network Monitor to describe sensor* | <blank> | Yes <br> (on Network Monitor host) |
| Group ID <br> *Network ID used to identify members of the same wireless network* | Pre-programmed <br> User re-programmable | Yes |
| Device ID <br> *Device ID used to uniquely identify each node in a wireless network* | Pre-programmed <br> User re-programmable | Yes |

* Persisted indicates value is retained after power cycle.

## End Node Setup (EN-5424)

The End Nodes, model number EN-5424, are mounted to terminal boards as shown in Figure 2-8. 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 End Node. The label on top of the End Node contains the device and group IDs assigned to the End Node.

**Figure 2-8. End node and terminal board (top and bottom views)**



To provide power to the End Node, a 3 VDC lithium coin cell (CR2450 type) is provided and already installed in the battery holder on the back of the terminal board.

To activate an end node:

- Turn the terminal board's power switch **ON** (refer to Figure 2-8). The End Node is ready to communicate directly or indirectly (through an Mesh Node) with the MeshGate. Repeat this step for each End Node.

***Note:*** Configuring and connecting an End Node or Mesh Node to external devices via their digital or analog I/O connectors is discussed in Chapter 3, “Running MeshScape Network Monitor”.

### Jumper settings

Note the following jumper settings on the end node terminal board:

- P6 - Install a jumper across P6 to enable the terminal board's LEDs as follows:
  - LED1 indicates power on/off.
  - LED12 indicates the end node has a primary parent.
  - LED2 indicates the end node has a secondary parent.
  - LED3 – 6 & 10,11,13,14 indicate the state of their respective port (For example, if the end node is configured in UART mode, LED4 will be active when receiving data (RX line).
- P14 is used to connect the RS-232 transceiver's I/O pins to the end node's UART. Jumpers must be engaged to use the end node's RS-232 interface. To use the RS-232 transceiver, it must be enabled by populating R6 with a 1K resistor and depopulating R19.

### Default settings

Table 2-7 lists the default settings for the MeshScape end node.

**Table 2-7. MeshScape end node default settings**

| **Variable** *Description* | **Default Value** | **Persisted?*** |
|---|---|---|
| Sampling Interval<br>*Defines the time interval between successive sensor samples sent from end node to MeshGate* | 10 Seconds | No |
| A/D Converter setup<br>*Defines state of A/D converter sensor inputs* | Channels 0-3 disabled | Yes |
| DIO Setup<br>*Defines state of digital I/O lines* | DIO<0:3> configured as inputs | Yes |
| Serial Data configuration<br>*Defines configuration of serial data sensor input, options are:*<br>***Off** - DIO<0:3> configured as I/O lines*<br>***Digital UART** - DIO<0:3> configured as digital UART signals* | **Off** - DIO<0:3> defined as I/O lines | Yes |
| Label<br>*Text label used to describe sensor* | <blank> | Yes<br>(on Network Monitor host) |
| Group ID<br>*Network ID used to identify members of the same wireless network* | Pre-programmed<br>User re-programmable | Yes |

**Table 2-7. MeshScape end node default settings (continued)**

| **Variable**<br>***Description*** | **Default Value** | **Persisted?*** |
|---|---|---|
| Device ID<br>*Device ID used to uniquely identify each node in a wireless network* | Pre-programmed<br>User re-programmable | Yes |

* Persisted indicates value is retained after power cycle.

# Installing MeshScape Network Monitor

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

1. Use the CD-ROM shipped with the reference kit to install the following Millennial Net items onto the host PC:
   - MeshScape Network Monitor
   - MeshScape Programmer
   - API software
   - EN-5424 End Node Tech Sheet
   - MG-5424 MeshGate Tech Sheet
   - MN-5424 Mesh Node Tech Sheet
   - RK-5424 MeshScape Users Guide
   - RK-5424-5 Kit Contents
   - IK-5424 Kit Contents
   - Reference Kits Tech Sheet
   - 5424 Family Product Sheet
   - 5424 MeshScape Release Notes
2. Open a MeshScape Network Monitor session.

The software installation procedure utilizes an InstallShield Wizard that will guide you through the installation process. When the process is complete, shortcut icons for MeshScape Network Monitor and MeshScape Programmer are also added to the host PC's desktop.

## Installing Contents of Millennial Net's RK-5424 CD-ROM

To install the software contained on the CD-ROM:

Insert the *RK-5424 Reference Kit CD* into the host PC's CD-ROM drive. The Autorun feature launches the InstallShield Wizard. Follow the prompts to install the contents of the CD onto the host PC.

If Autorun is not enabled, drill down to the contents of the kit CD and double-click on **setup.exe**. The InstallShield Wizard is launched. Follow the prompts to install the contents of the CD onto the host PC.

***Note:*** If a version of MeshScape Network Monitor already exists on the host PC, it will be detected during the installation process. A special prompt screen is then displayed, allowing you to remove the existing files. You must select **Remove** to unistall the existing version before running the installation program again to install the newer version contained on the CD.

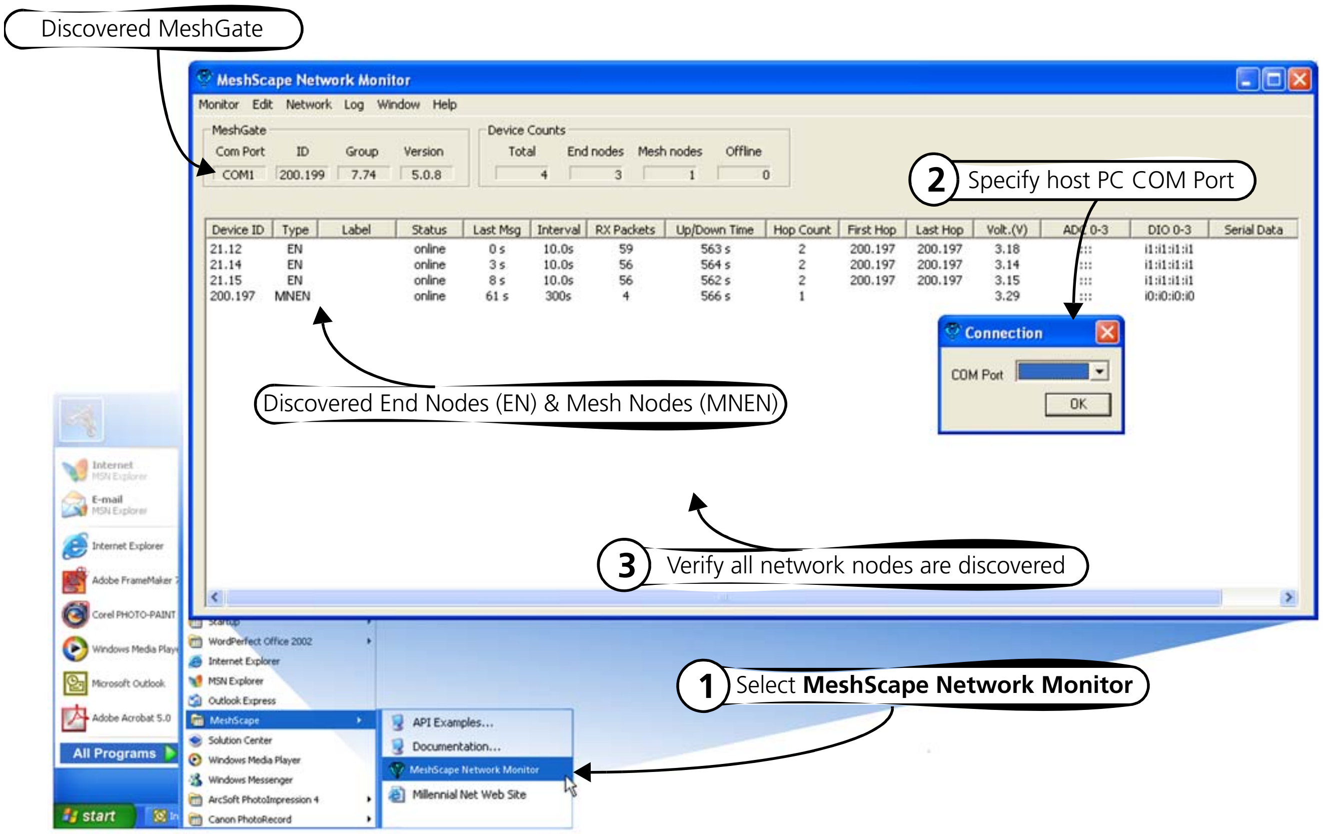
Proceed to 'Launching MeshScape Network Monitor Using Windows' on page 2-16.

## Launching MeshScape Network Monitor Using Windows

Using the standard application launching methods of Windows, the following procedure describes how to launch MeshScape Network Monitor and verify proper communication with the network nodes (see Figure 2-9):

1. To launch MeshScape Network Monitor, do one of the following:
   - Double-click on the desktop's MeshScape Network Monitor icon.
   - From the Windows taskbar, select:
     **Start>All Programs>MeshScape>MeshScape Network Monitor**.
2. Enter the COM port on which the host PC is to communicate with the connected MeshScape.
3. Verify that all network nodes are discovered and displayed by MeshScape Network Monitor.

**Figure 2-9. Using Windows Start menu to launch MeshScape Network Monitor**


Discovered MeshGate
MeshScape Network Monitor
Monitor Edit Network Log Window Help
2 Specify host PC COM Port
Connection
COM Port
OK
Discovered End Nodes (EN) & Mesh Nodes (MNEN)
3 Verify all network nodes are discovered
1 Select **MeshScape Network Monitor**
All Programs
start
WordPerfect Office 2002
Internet Explorer
MSN Explorer
Outlook Express
MeshScape
Solution Center
Windows Media Player
Windows Messenger
ArcSoft PhotoImpression 4
Canon PhotoRecord
API Examples...
Documentation...
MeshScape Network Monitor
Millennial Net Web Site

MeshGate

| Com Port | ID | Group | Version |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5.0.8 |

Device Counts

| Total | End nodes | Mesh nodes | Offline |
|---|---|---|---|
| 4 | 3 | 1 | 0 |

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 | DIO 0-3 | Serial Data |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 0 s | 10.0s | 59 | 563 s | 2 | 200.197 | 200.197 | 3.18 | ::: | i1:i1:i1:i1 | |
| 21.14 | EN | | online | 3 s | 10.0s | 56 | 564 s | 2 | 200.197 | 200.197 | 3.14 | ::: | i1:i1:i1:i1 | |
| 21.15 | EN | | online | 8 s | 10.0s | 56 | 562 s | 2 | 200.197 | 200.197 | 3.15 | ::: | i1:i1:i1:i1 | |
| 200.197 | MNEN | | online | 61 s | 300s | 4 | 566 s | 1 | | | 3.29 | ::: | i0:i0:i0:i0 | |

Once proper operation of the MeshScape system has been verified, proceed to Chapter 3, "Running MeshScape Network Monitor" for an overview of the GUI and details on how to use it to configure the operation of your MeshScape system.

# 3

# Running MeshScape Network Monitor

This chapter provides the following MeshScape Network Monitor information:

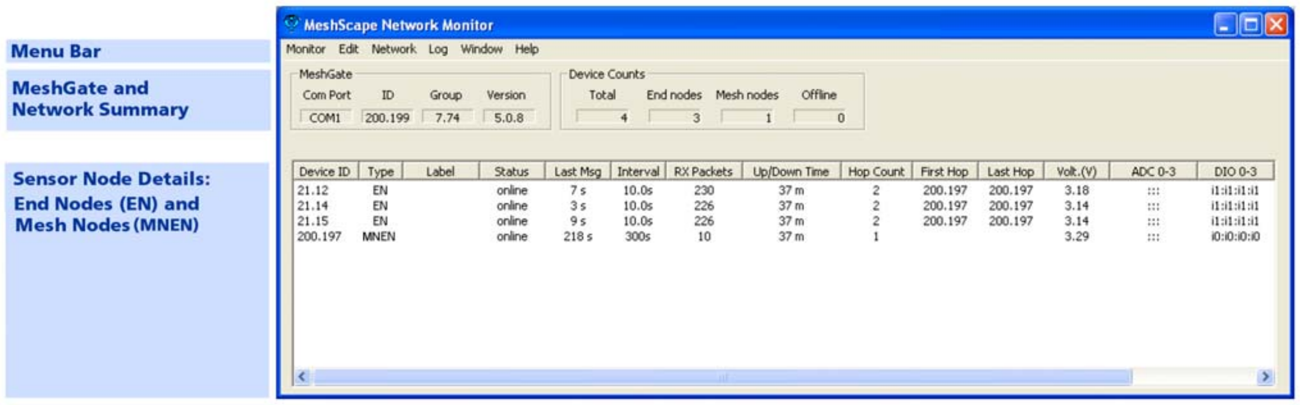
# MeshScape Network Monitor Overview

Millennial Net’s MeshScape Network Monitor is a monitoring and management system for MeshScape networks. This management tool will discover and display active Mesh Nodes and End Nodes in range of the MeshGate as shown in Figure 3-10. MeshScape Network Monitor displays the *Group ID* and *Device ID* of the MeshGate and will display only End Nodes and Mesh Nodes that have the same group ID as the MeshGate. (For information on opening a MeshScape Network Monitor session, see ‘Launching MeshScape Network Monitor Using Windows’ on page 2-16.)

Using MeshScape Network Monitor, a number of the monitoring features may be observed:

- **Any of the Mesh Nodes can be moved, and as long as they are within the range of a Mesh Node or the MeshGate, connectivity will be maintained seamlessly.** Any of the Mesh Nodes and even the MeshGate can be moved while operating within range, and all routes will automatically adapt to their new locations.
- **The MeshScape system Persistent Dynamic Routing™ routing protocol always seeks to route data using the most reliable RF links with 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 MeshScape Network Monitor. For example, the Mesh Node IDs used for the first and last hops may change from time to time even when the End Node is stationary, due to environmental interference.
- **If any of the Mesh Nodes runs out of battery power or is turned off, all routes that went through that Mesh Node will be reconfigured—all End Nodes communicating with that Mesh Node will still be connected to other Mesh Nodes without any disruption or loss of packets.** However, if an End Node exceeds the range of the network due to the loss of an Mesh Node, then the End Node will be displayed as Offline.

**Figure 3-10. Sample MeshScape Network Monitor window**


Menu Bar
MeshGate and Network Summary
Sensor Node Details: End Nodes (EN) and Mesh Nodes (MNEN)
MeshScape Network Monitor
Monitor
Edit
Network
Log
Window
Help
MeshGate
Device Counts

| Com Port | ID | Group | Version |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5.0.8 |

| Total | End nodes | Mesh nodes | Offline |
|---|---|---|---|
| 4 | 3 | 1 | 0 |

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 | DIO 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 7 s | 10.0s | 230 | 37 m | 2 | 200.197 | 200.197 | 3.18 | ::: | i1:i1:i1:i1 |
| 21.14 | EN | | online | 3 s | 10.0s | 226 | 37 m | 2 | 200.197 | 200.197 | 3.14 | ::: | i1:i1:i1:i1 |
| 21.15 | EN | | online | 9 s | 10.0s | 226 | 37 m | 2 | 200.197 | 200.197 | 3.14 | ::: | i1:i1:i1:i1 |
| 200.197 | MNEN | | online | 218 s | 300s | 10 | 37 m | 1 | | | 3.29 | ::: | i0:i0:i0:i0 |

As shown in Figure 3-10, 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:

  – **Exit**: Ends the session and closes MeshScape Network Monitor.

- **Edit**

  – **Labels:** Assign user-defined names to End Nodes and Mesh Nodes on the network. For details, see 'Labeling an End Node or Mesh Node' on page 3-18.

  – **Persistence:** Toggle monitoring/displaying offline End Nodes and Mesh Nodes. For details, see 'Configuring Persistence Attributes' on page 3-19

  – **Data Format**: 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 'Configuring Serial and ADC Data Formats' on page 3-21.

- **Network**

  – **Connection**: Select serial port on Host PC to use for MeshGate connection. For details, see 'Selecting a Com Port on the Host PC' on page 3-20.

  – **All Sampling Intervals**: Configure all network nodes with the same sampling interval time. For details, see 'Configuring the Sample Interval of all Network Nodes' on page 3-9.

  – **Events**: Turn event tracking on or off. Tracked events are included in the Event log file. For details, see 'Turning Event Tracking On/Off' on page 3-22.

  – **Broadcast**: Broadcast data to all nodes on your MeshScape system including the time to which all nodes will synchronize their clocks and Ultra Low Power (ULP) settings defining wakeup interval and duty cycle for all nodes. For details, see 'Broadcasting Data to All Nodes.' on page 3-23. The broadcast feature is not supported in this release of the RK-5424-5 Reference Kit but will be supported in future releases.

- **Log**

  – **Attributes**: 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 3-25.

  – **View**: Display contents of log file. For details, see 'Viewing the Contents of an Event Log File' on page 3-26.

- **Window**

  – **Watch:** Display the current status information relating to a node's interfaces. For details, see 'Using Watch Function to Display Configuration Information' on page 3-16.

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

- **Help**

  – **About**: Displays MeshScape Network Monitor revision level information.

## MeshGate

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

- **Com Port**: Host PC's RS-232 port connected to MeshGate.
- **ID**: Device identifier assigned to the MeshGate. The ID consists of two octets (A.B), where each octet's value = 160 to 255.
- **Group**: Group identifier assigned to the MeshGate. All Mesh Nodes and End Nodes with the same Group identifier will communicate with the displayed MeshGate.
- **Version**: Version of firmware loaded on the MeshGate.

## Device Counts

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

- **Total**: Total combined number of discovered End Nodes and Mesh Nodes.
- **End nodes**: Total number of End Nodes currently online.
- **Mesh nodes**: Total number of discovered Mesh Nodes currently online.
- **Offline:** Total number of discovered end nodes and mesh nodes currently offline.

## Sensor Node Details

This section displays the following information related to the End Nodes and Mesh Nodes on the network:

- **Device ID**: Unique identifier assigned each node. The identifier consists of two octets (A.B), where each octet contains a value between 001 and 255.
- **Type**: This column lists all nodes discovered on the network that are assigned the same group ID as the displayed MeshGate. Nodes displayed here include End Nodes (EN) and Mesh Nodes (MNEN) or (MN) for Mesh Nodes without I/O capabilities.
- **Label**: User-defined name assigned to node via MeshScape Network Monitor application.
- **Status**: Current status of the device:
  - **Online**: The node is communicating with the MeshGate.
  - **Offline**: The MeshGate can no longer communicate with the node.
  - **Refresh**: The node is being updated with a new operating state.
- **Last Msg**: Time elapsed since last packet was received from or transmitted to 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).

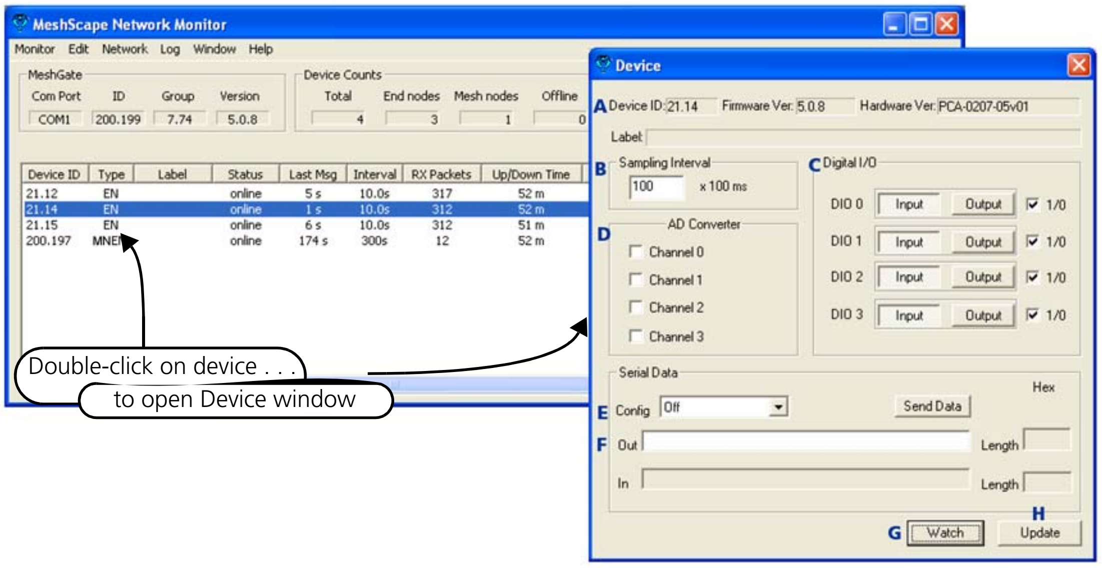
- **RX Packets**: Number of packets successfully delivered to MeshScape Network Monitor from a node since the node was detected by the MeshGate. The counter is reset if the MeshScape Network Monitor program is restarted.
- **Up/Down Time**: Total time since the node was last detected on/off by the MeshGate.
- **Hop Count**: Number of network node hops taken by a packet delivered from a node to the MeshGate. For example: End Node—MeshGate = 1 hop,
  End Node—Mesh Node—MeshGate = 2 hops (each additional Mesh Node will add another hop).
- **First Hop**: Device ID of the first Mesh Node on the path used by a packet to get to the MeshGate. If no Mesh Node was used, then the field is blank, indicating the device is communicating directly with the MeshGate.
- **Last Hop**: Device ID of the last Mesh Node on the path used by a packet to get to the MeshGate. If no Mesh Node was used, then the field is blank, indicating the device is communicating directly with the MeshGate.
- **Volt**: DC voltage level of the node's power source in volts.
- **ADC 0-3 Data**: Input voltages on pins used for analog-to-digital conversion operation.
- **DIO 0-3 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.

## Configuring a Node’s Operation

To configure the operation of an End Node or Mesh Node, double-click on the desired device from the list of discovered sensor nodes in the main display. The Device window is displayed, showing the current configuration of the selected device (see Figure 3-11).

The Device window is a modeless dialog that may be left open while you interact with other MeshScape Network Monitor features, windows, and dialogs.

**Figure 3-11. MeshScape Network Monitor’s Device window**


MeshScape Network Monitor
Monitor Edit Network Log Window Help
Device
A
B
C
D
E
F
G
H
Watch
Update
Send Data
Input
Output
Double-click on device . . .
to open Device window

MeshGate

| Com Port | ID | Group | Version |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5.0.8 |

Device Counts

| Total | End nodes | Mesh nodes | Offline |
|---|---|---|---|
| 4 | 3 | 1 | 0 |

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time |
|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 5 s | 10.0s | 317 | 52 m |
| 21.14 | EN | | online | 1 s | 10.0s | 312 | 52 m |
| 21.15 | EN | | online | 6 s | 10.0s | 312 | 51 m |
| 200.197 | MNE | | online | 174 s | 300s | 12 | 52 m |

Device ID: 21.14 Firmware Ver: 5.0.8 Hardware Ver: PCA-0207-05v01

Label:

Sampling Interval: 100 x 100 ms

AD Converter: ☐ Channel 0 ☐ Channel 1 ☐ Channel 2 ☐ Channel 3

Digital I/O: DIO 0 ☑ 1/0; DIO 1 ☑ 1/0; DIO 2 ☑ 1/0; DIO 3 ☑ 1/0

Serial Data: Config: Off; Hex; Out; Length; In; Length

Table 3-8 describes the functions of the various sections of the window as shown in Figure 3-11.

**Table 3-8. Device window functions**

| Item | Description | Function |
|---|---|---|
| **A** | Device ID | This is the device ID of the node currently selected for configuring. |
| **B** | Sampling Interval | This setting configures how often the node transmits a ‘heart beat’ data packet or any other data.<br>For details, see ‘Configuring the Sample Interval of a Single Node’ on page 3-8. |
| **C** | Digital I/O | This panel is used to control the states of the I/O pins associated with digital I/O channels D0–D3. 1 = High; 0 = LOW.<br>For details, see ‘Configuring Digital I/O Operation’ on page 3-9 |
| **D** | AD Converter | This panel is used to control the states of the AD (Analog-to-Digital) Converter channels.<br>For details, see ‘Configuring AD (analog-to-digital) Converter Operation’ on page 3-14. |

**Table 3-8. Device window functions (continued)**

| Item | Description | Function |
|---|---|---|
| **E** | Serial Data Config | This panel is used to select a serial I/O operation for a device: Off, Digital UART, RS-232 (not supported), or RS-485 (not supported). Selecting serial operation disables digital I/O functionality because these operations share the same pins.<br>For details, see the following:<br>• ’Configuring UART Operation’ on page 3-12 |
| **F** | Serial Data Out | This panel, which is only used if the node is configured for serial operation, is used to send serial data to the node.<br>For details, see the following:<br>• ’Configuring UART Operation’ on page 3-12 |
| **G** | Watch | This option opens a new window that displays information relating to the node’s various interfaces, including analog and digital I/O configuration states and packets received/sent.<br>Use this window to observe when configured changes have actually occurred on the device.<br>For details, see ’Using Watch Function to Display Configuration Information’ on page 3-16 |
| **H** | Update | Updates the selected device with any changes made to its configuration. Updating a device will reset the “last message” count. |

## Configuring the Sample Interval of a Single Node

To configure the time interval between data packets transmitted by a node (see Figure 3-12):

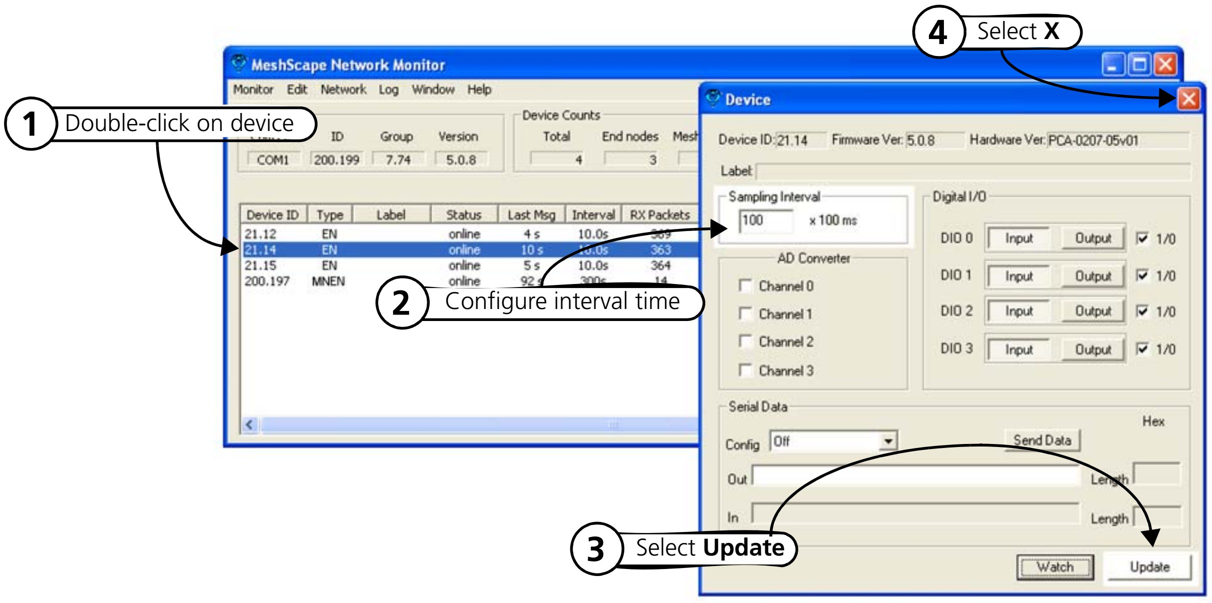
1. Double-click on the desired device from the list of discovered sensor nodes. The Device window is opened, displaying the device's current configuration.
2. Using the *Sampling Interval* panel, enter the interval sampling rate as a multiple of 100 milliseconds. For example, to configure a sampling interval rate of 10 seconds, enter a value of 100.

   *minimum value*: 1 (0.1 sec)
   *maximum value*: 65535 (109 minutes)
3. Select **Update**. The device's configuration is updated.

**Note:** After you click Update, the MeshScape Network Monitor will display the 'Refresh' state for the device. Changes to the device configuration will show up after one sampling interval. You can use the Watch window to track configuration changes.

4. Select **X** to exit the Device window.

**Figure 3-12. Configuring sample interval of single node**


4 Select **X**
MeshScape Network Monitor
Monitor Edit Network Log Window Help
1 Double-click on device
Device
2 Configure interval time
3 Select **Update**
Send Data
Watch
Update

| | ID | Group | Version |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5.0.8 |

Device Counts: Total 4, End nodes 3

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets |
|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 4 s | 10.0s | 369 |
| 21.14 | EN | | online | 10 s | 10.0s | 363 |
| 21.15 | EN | | online | 5 s | 10.0s | 364 |
| 200.197 | MNEN | | online | 92 s | 300s | 14 |

Device ID: 21.14 Firmware Ver: 5.0.8 Hardware Ver: PCA-0207-05v01

Label:

Sampling Interval: 100 x 100 ms

AD Converter: ☐ Channel 0 ☐ Channel 1 ☐ Channel 2 ☐ Channel 3

Digital I/O: DIO 0 Input Output ☑ 1/0; DIO 1 Input Output ☑ 1/0; DIO 2 Input Output ☑ 1/0; DIO 3 Input Output ☑ 1/0

Serial Data: Config Off; Out, Length; In, Length; Hex

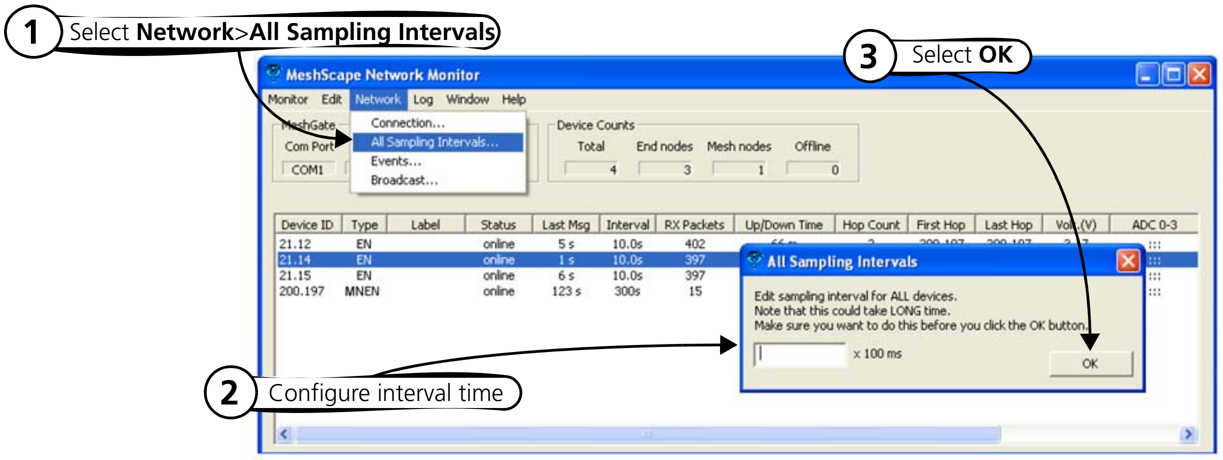
## Configuring the Sample Interval of all Network Nodes

***Note:*** It can take a long time to change the sampling interval for all network nodes.

To configure all the network nodes with the same sampling interval rate (see Figure 3-13):

1. Select **Network**>**All Sampling Intervals**. The Edit Sampling Interval window is opened.
2. Enter the interval rate as a multiple of 100 milliseconds. For example, to configure a sampling interval rate of 20 seconds, enter a value of 200.
3. Select **OK**. All nodes on the network are configured with the sampling interval rate entered.

**Figure 3-13. Configuring sample interval of all nodes**


1 Select **Network**>**All Sampling Intervals**
3 Select **OK**
MeshScape Network Monitor
Monitor Edit Network Log Window Help
Connection...
All Sampling Intervals...
Events...
Broadcast...
MeshGate
Com Port
COM1
Device Counts
Total 4
End nodes 3
Mesh nodes 1
Offline 0
All Sampling Intervals
x 100 ms
OK
2 Configure interval time

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 5 s | 10.0s | 402 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | ::: |
| 21.14 | EN | | online | 1 s | 10.0s | 397 | | | | | | ::: |
| 21.15 | EN | | online | 6 s | 10.0s | 397 | | | | | | ::: |
| 200.197 | MNEN | | online | 123 s | 300s | 15 | | | | | | ::: |

Edit sampling interval for ALL devices.
Note that this could take LONG time.
Make sure you want to do this before you click the OK button.

## Configuring Digital I/O Operation

The following procedure describes the steps that need to be taken to set up the hardware and configure an End Node or Mesh Node for digital I/O operation (see Figure 3-14).

***Note:*** All Digital I/O contain on-board pull-up resistors. To optimize current consumption, be sure to take into account the pullups when setting I/O values.

### Digital Input Setup

1. Connect the digital source signals to the connectors (DIO 0–DIO 3) and Ground (GND) of:
   – End Node's digital terminal block (P9 located on terminal board).
   – Mesh Node's terminal block pins.

   See Figure 3-14 on page 3-11 for connector locations.
2. From MeshScape Network Monitor, double-click on the desired device from the list of discovered sensor nodes. The Device window is opened, displaying the device's current configuration.

3. 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**.

   MeshScape Network Monitor displays the digital information for the node in the Digital I/O Data column, where **On** = output channel and **In** = input channel (n = 1 or 0).

### Digital Output Setup

1. Connect the digital signal destination devices to the connectors (DIO 0–DIO 3) and ground (GND) of the following:

   – End Node's terminal block (**P9** located on terminal board).

     Pin 1 - Gnd

     Pin 2 - DIO 0 RXD (Output)

     Pin 3 - DIO 1 TXD (Input)

     Pin 4 - DIO 2 RTS (Input)

     Pin 5 - DIO 3 CTS (Output)

   – Mesh Node's terminal block connector.

   See Figure 3-14 on page 3-11 for connector locations.

2. From MeshScape Network Monitor, double-click on the desired device from the list of discovered sensor nodes. The Device window is opened, displaying the device's current configuration.

3. From the *Digital I/O* panel, select **Output** on each of the desired digital channels to use as outputs (DIO 0—DIO 3).

4. 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)

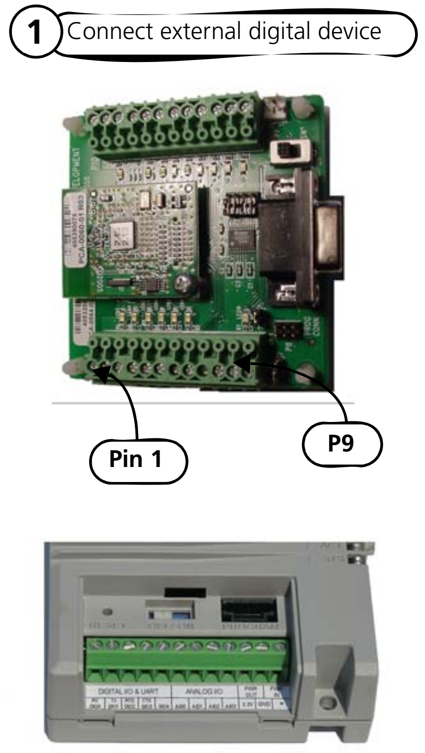
   – Not selected = 0 (output is set low)

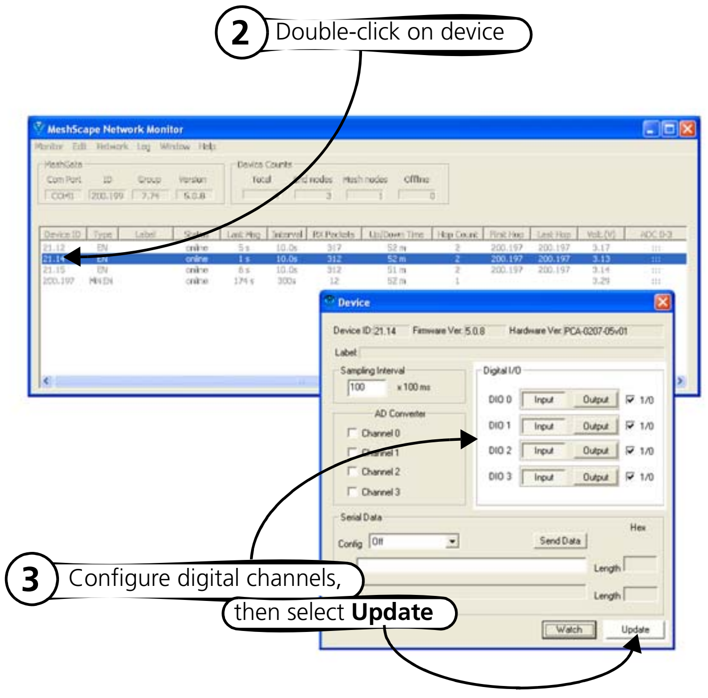
5. Select **Update**.

   MeshScape Network Monitor displays the digital information for the node in the Digital I/O Data column, where **On** = output channel and **In** = input channel (n = 1 or 0).

***Note:*** Input signals should not be applied when the end 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 3-14. Configuring End Node or mesh node for digital I/O**





| DIGITAL I/O & UART | | | | | ANALOG I/O | | | | PWR OUT | | PWR IN |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RX | TX | RTS | CTS | | | | | | | | |
| DIO0 | DIO1 | DIO2 | DIO3 | DIO4 | A/D0 | A/D1 | A/D2 | A/D3 | 3.3V | GND | + |

## Configuring UART Operation

The following procedures describes how to use the End Node's digital I/O connections or the Mesh Node's UART connector for serial/UART communications (see Figure 3-15). Refer to the technical specification sheets for the End Node and Mesh Node for additional information required when using the device for serial communications.

***Note:*** The End Node supports the following serial communication parameters:
9600 bps, 8 data bits, no parity, 1 stop bits

The mesh node supports the following serial communication parameters:
9600 bps, 8 data bits, no parity, 2 stop bits

Serial data and digital I/O are mutually exclusive operations for end nodes.

1. Connect to the following:
   - End Node's **P9** terminal block (pins DIO 0–DIO 3, GND).

     Pin 1 - Gnd

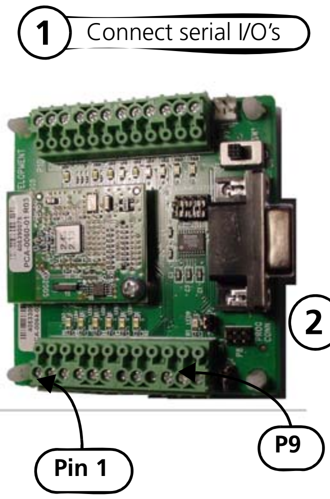
     Pin 2 - DIO 0 RXD (Output)

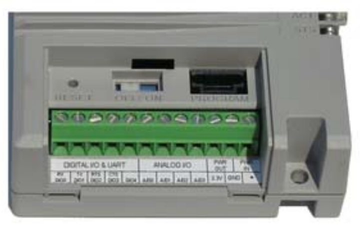
     Pin 3 - DIO 1 TXD (Input)

     Pin 4 - DIO 2 RTS (Input)

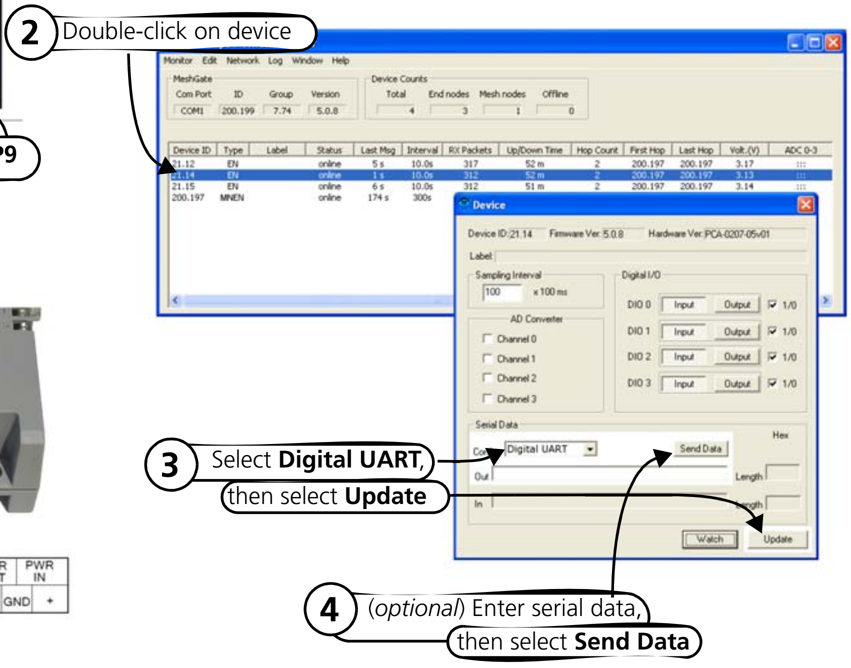
     Pin 5 - DIO 3 CTS (Output)
   - Mesh Node's terminal block connector (pins DIO 0-DIO 3, GND).
2. From MeshScape Network Monitor, double-click on the desired device from the list of discovered sensor nodes. The Device window is opened, displaying the device's current configuration.
3. From the *Config* panel, select Serial Data: **Digital UART**, then **Update**. The UART terminal block is ready for UART operation (digital I/O function is disabled).
4. (*optional*) To send serial data (HEX, DEC, or ASCII) to the node, enter the data in the *Out* panel, then select **Send Data**.

**Figure 3-15. Configuring End Node or mesh node for UART operation**





| DIGITAL I/O & UART | | | | | ANALOG I/O | | | | PWR OUT | PWR IN | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RX | TX | RTS | CTS | | | | | | | | |
| DIO0 | DIO1 | DIO2 | DIO3 | DIO4 | A/D0 | A/D1 | A/D2 | A/D3 | 3.3V | GND | + |


2 Double-click on device
3 Select **Digital UART**, then select **Update**
4 (*optional*) Enter serial data, then select **Send Data**
Monitor Edit Network Log Window Help
Device
Send Data
Watch
Update

MeshGate: Com Port COM1; ID 200.199; Group 7.74; Version 5.0.8

Device Counts: Total 4; End nodes 3; Mesh nodes 1; Offline 0

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 5 s | 10.0s | 317 | 52 m | 2 | 200.197 | 200.197 | 3.17 | ::: |
| 21.14 | EN | | online | 1 s | 10.0s | 312 | 52 m | 2 | 200.197 | 200.197 | 3.13 | ::: |
| 21.15 | EN | | online | 6 s | 10.0s | 312 | 51 m | 2 | 200.197 | 200.197 | 3.14 | ::: |
| 200.197 | MNEN | | online | 174 s | 300s | | | | | | | |

Device ID: 21.14 Firmware Ver: 5.0.8 Hardware Ver: PCA-0207-05v01

Label:

Sampling Interval: 100 x 100 ms

AD Converter: ☐ Channel 0 ☐ Channel 1 ☐ Channel 2 ☐ Channel 3

Digital I/O:
DIO 0 Input Output ☑ 1/0
DIO 1 Input Output ☑ 1/0
DIO 2 Input Output ☑ 1/0
DIO 3 Input Output ☑ 1/0

Serial Data: Digital UART; Out; In; Hex; Length; Length

## 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 End Node or Mesh Node for AD Converter operation (see Figure 3-16):

***Note:*** The input range of the analog inputs is 0 to $V_{DD}$ (power supply voltage).

1. Connect 0–3 VDC signals to the following A/D Converter connectors:
   - End Node: Connect signal source ground to GND_CPU (pin 1 on terminal block **P10**).

     Pin 1 - Gnd

     Pin 2 - AD0 (Input)

     Pin 3 - AD1 (Input)

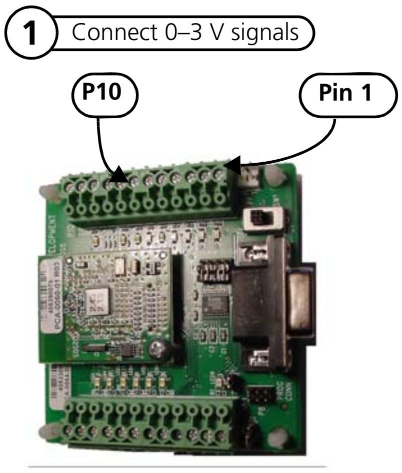
     Pin 4 - AD2 (Input)

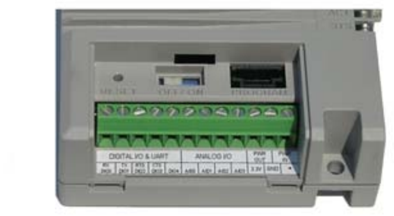
     Pin 5 - AD3 (Input)
   - Mesh Node: Connect signal source ground to terminal block connector Pin 11 (Ground).

   Use any or all of the A/D Converter channels (AD0–AD3) for connecting analog signal sources.
2. From MeshScape Network Monitor, double-click on the desired device from the list of discovered sensor nodes. The Device window is opened, displaying the device’s current configuration.
3. From the *AD Converter* panel, select the AD Converter channels to which the external analog signals have been connected, then select **Update**.

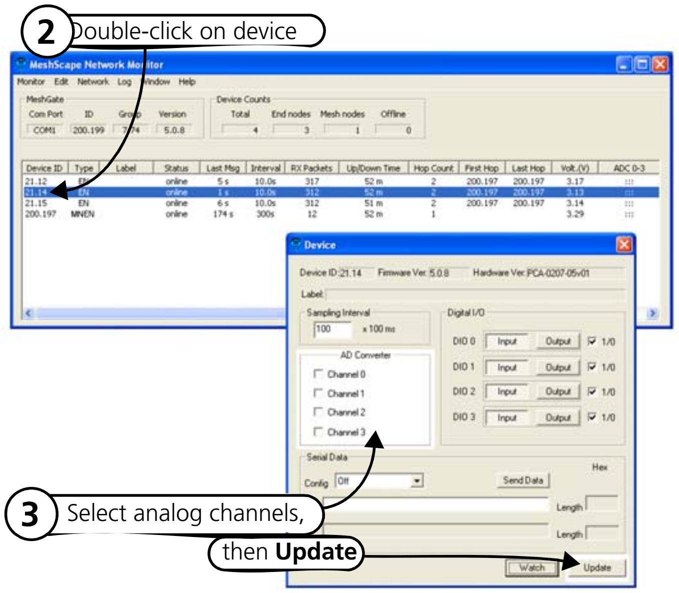
   MeshScape Network Monitor displays the analog information for the node in the ADC0-3 column.

**Figure 3-16. Configuring End Node/Mesh Node for analog I/O**





| DIGITAL I/O & UART | | | | | ANALOG I/O | | | | PWR OUT | PWR IN | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RX | TX | RTS | CTS | | | | | | | | |
| DIO0 | DIO1 | DIO2 | DIO3 | DIO4 | A/D0 | A/D1 | A/D2 | A/D3 | 3.3V | GND | + |



MeshGate: Com Port COM1, ID 200.199, Group 7774, Version 5.0.8

Device Counts: Total 4, End nodes 3, Mesh nodes 1, Offline 0

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 5 s | 10.0s | 317 | 52 m | 2 | 200.197 | 200.197 | 3.17 | ::: |
| 21.14 | EN | | online | 1 s | 10.0s | 312 | 52 m | 2 | 200.197 | 200.197 | 3.13 | ::: |
| 21.15 | EN | | online | 6 s | 10.0s | 312 | 51 m | 2 | 200.197 | 200.197 | 3.14 | ::: |
| 200.197 | MNEN | | online | 174 s | 300s | 12 | 52 m | 1 | | | 3.29 | ::: |

Device ID: 21.14 Firmware Ver: 5.0.8 Hardware Ver: PCA-0207-05v01

Label:

Sampling Interval: 100 x 100 ms

AD Converter: ☐ Channel 0 ☐ Channel 1 ☐ Channel 2 ☐ Channel 3

Digital I/O:

| | | | |
|---|---|---|---|
| DIO 0 | Input | Output | ☑ 1/0 |
| DIO 1 | Input | Output | ☑ 1/0 |
| DIO 2 | Input | Output | ☑ 1/0 |
| DIO 3 | Input | Output | ☑ 1/0 |

Serial Data: Config Off, Hex, Length, Length

## Using Watch Function to Display Configuration Information

Use the Watch window to track changes to a device's configuration.

You may open multiple instances of the Watch window. The selected device's ID and label (if configured) are displayed in the Watch window title bar.

To display the current and desired configuration for a device (see Figure 3-17):

1. Double-click on the desired device from the list of discovered sensor nodes. The Device window is opened, displaying the device's current configuration.
2. Select **Watch**. The **Watch** window is opened, displaying the selected device's current and desired configuration information.
3. Select **X** to exit the Watch window.

Alternatively, click once on a device and then select **Window>Watch** to open the Watch window for that device.

**Figure 3-17. Displaying I/O information using Watch function**

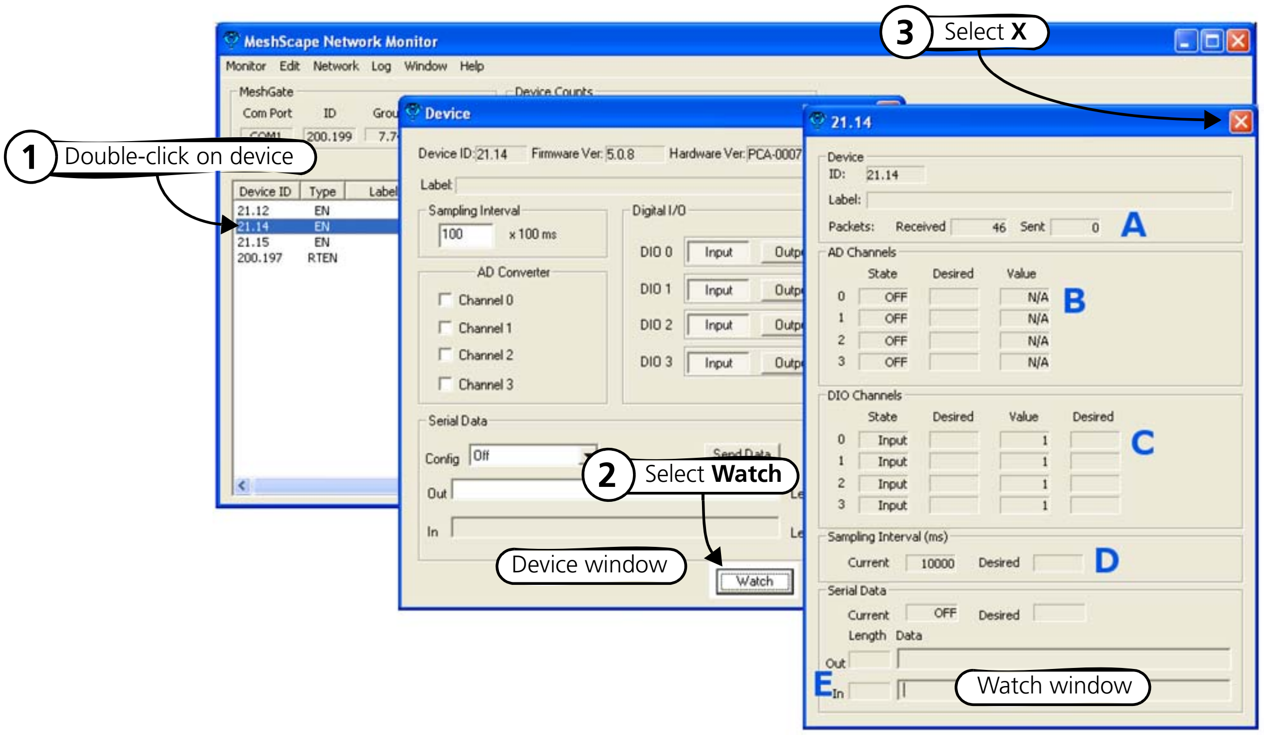


Table 3-9 describes the functions of the various sections of the Watch window as shown in Figure 3-17.

**Table 3-9. Watch window functions**

| Item | Description | Function |
|---|---|---|
| **A** | Packets Received/Sent | Total packets received from or transmitted to the node. |
| **B** | A-D Channels | This panel displays the following information for each analog-to-digital channel (0–3):<br>• **State**: Current on/off state of the channel<br>• **Desired**: Desired on/off state of channel, which can be changed using the **Device** window.<br>• **Value**: Numeric values of the channel. |
| **C** | DIO Channels | This panel displays the following information for each digital channel (0–3):<br>• **State**: Current input/output state of the channel<br>• **Desired**: Desired input/output state of channel, which can be changed using the **Device** window.<br>• **Value**: I/O values of the channel (1 or 0).<br>• **Desired**: Desired I/O value of channel. |
| **D** | Sampling Interval | This panel displays the following sampling interval information:<br>• **Current**: Currently configured setting.<br>• **Desired**: Desired setting, which can be changed using the **Device** window (see ’Configuring the Sample Interval of a Single Node’ on page 3-8). |
| **E** | Serial Data | This panel displays the following serial data information of data received from the node (In) or transmitted to the node (Out):<br>• **Current**: Currently configured setting (Off, RS-232, RS-485)<br>• **Desired**: Desired setting<br>• **Length**: Length of string received/transmitted. |

# Labeling an End Node or Mesh Node

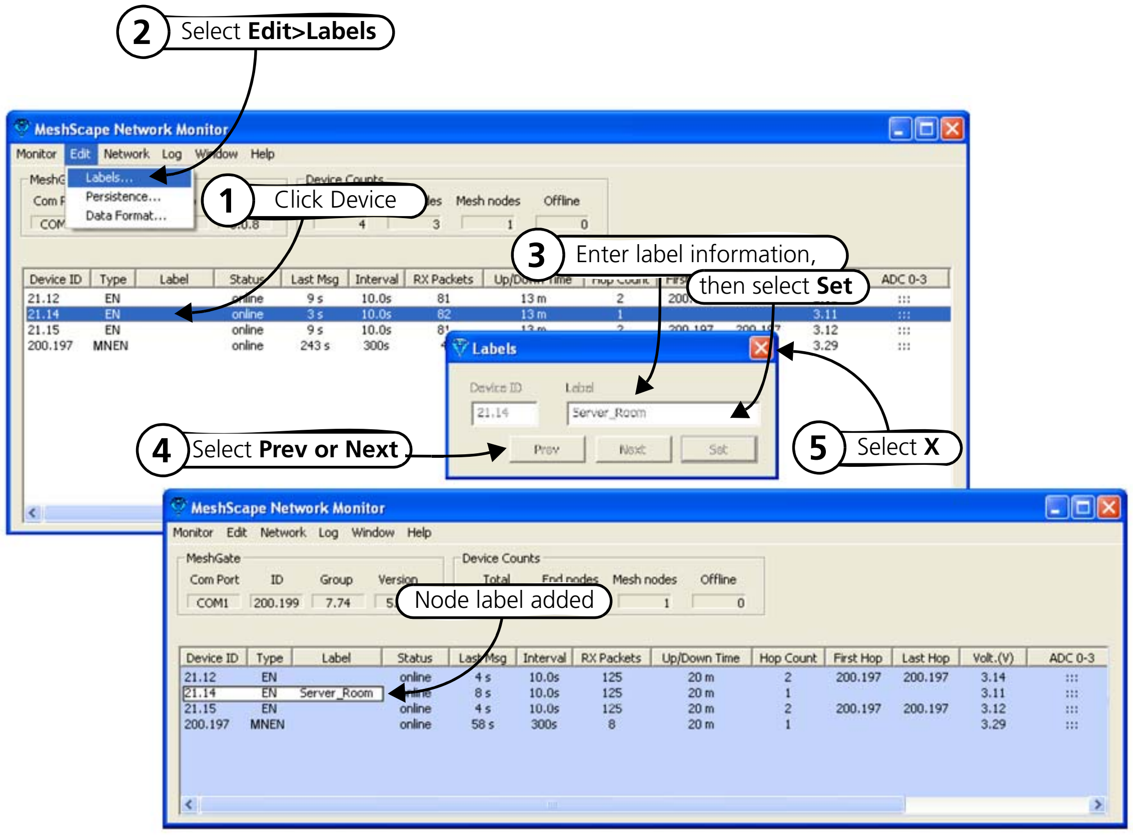
Labels are meaningful, persistent text strings used to identify network nodes beyond their group and device IDs.

**Note:** Labels are only accessible through the MeshScape Network Monitor program. Custom Applications using the API will not be able to access labels.

The following procedure describes the steps required to assign a label to an End Node or Mesh Node (see Figure 3-18):

1. From MeshScape Network Monitor, click on the desired device from the list of discovered sensor nodes.
2. From MeshScape Network Monitor, select **Edit**>**Labels**. The Edit Labels window is opened.
3. Enter the label to be applied to the node, then select **Set**. MeshScape Network Monitor displays the label in the *Label* column of the selected node.
4. Click **Prev** or **Next** to apply labels to other discovered sensor nodes.
5. Select **X** to exit the Edit Label window.

**Figure 3-18. Labeling an End Node or Mesh Node**



MeshGate

| Com Port | ID | Group | Version |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5. |

Device Counts

| Total | End nodes | Mesh nodes | Offline |
|---|---|---|---|
| | | 1 | 0 |

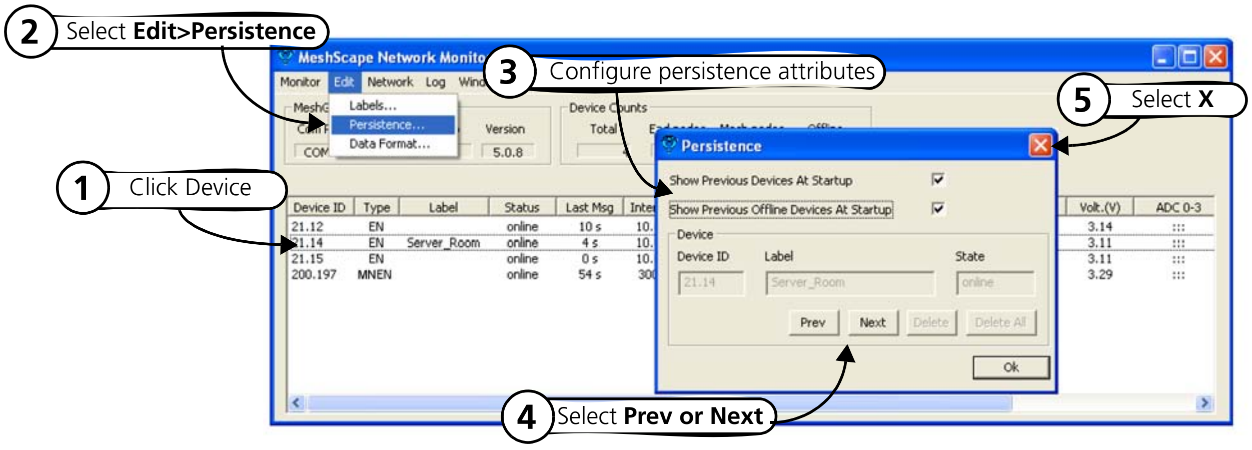
| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 4 s | 10.0s | 125 | 20 m | 2 | 200.197 | 200.197 | 3.14 | ::: |
| 21.14 | EN | Server_Room | online | 8 s | 10.0s | 125 | 20 m | 1 | | | 3.11 | ::: |
| 21.15 | EN | | online | 4 s | 10.0s | 125 | 20 m | 2 | 200.197 | 200.197 | 3.12 | ::: |
| 200.197 | MNEN | | online | 58 s | 300s | 8 | 20 m | 1 | | | 3.29 | ::: |

# Configuring Persistence Attributes

The following procedure describes how to configure the persistence setting of the network's End Nodes and Mesh Nodes (see Figure 3-19):

1. From MeshScape Network Monitor, click on the desired device from the list of discovered sensor nodes.
2. From MeshScape Network Monitor, select **Edit**>**Persistence**. The Edit Persistence window is opened.
3. Configure the following global persistence attributes:
   - **Show previous devices at startup**: If selected, all online nodes will be displayed when MeshScape Network Monitor is restarted.
   - **Show previous offline devices at startup**: If selected, all online and offline nodes will be displayed when MeshScape Network Monitor is restarted.
4. Click **Prev** or **Next** to display MeshScape devices and their states. You can manually delete devices that are offline as follows:
   - **Delete**: Select the delete button to stop displaying the selected Offline node.
   - **Delete all**: Select this delete button to stop displaying all nodes with a status of Offline.
5. Select **X** to exit the Edit Persistence window.

**Figure 3-19. Configuring node persistence attributes**


2 Select **Edit>Persistence**
MeshScape Network Monitor
Monitor Edit Network Log Wind
3 Configure persistence attributes
5 Select **X**
Labels...
Persistence...
Data Format...
Version
5.0.8
Device Counts
Total
1 Click Device
Persistence
Prev
Next
Delete
Delete All
Ok
4 Select **Prev or Next**

| Device ID | Type | Label | Status | Last Msg | Inte | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 10 s | 10. | 3.14 | ::: |
| 21.14 | EN | Server_Room | online | 4 s | 10. | 3.11 | ::: |
| 21.15 | EN | | online | 0 s | 10. | 3.11 | ::: |
| 200.197 | MNEN | | online | 54 s | 30 | 3.29 | ::: |

Show Previous Devices At Startup ✓
Show Previous Offline Devices At Startup ✓

Device

| Device ID | Label | State |
|---|---|---|
| 21.14 | Server_Room | online |

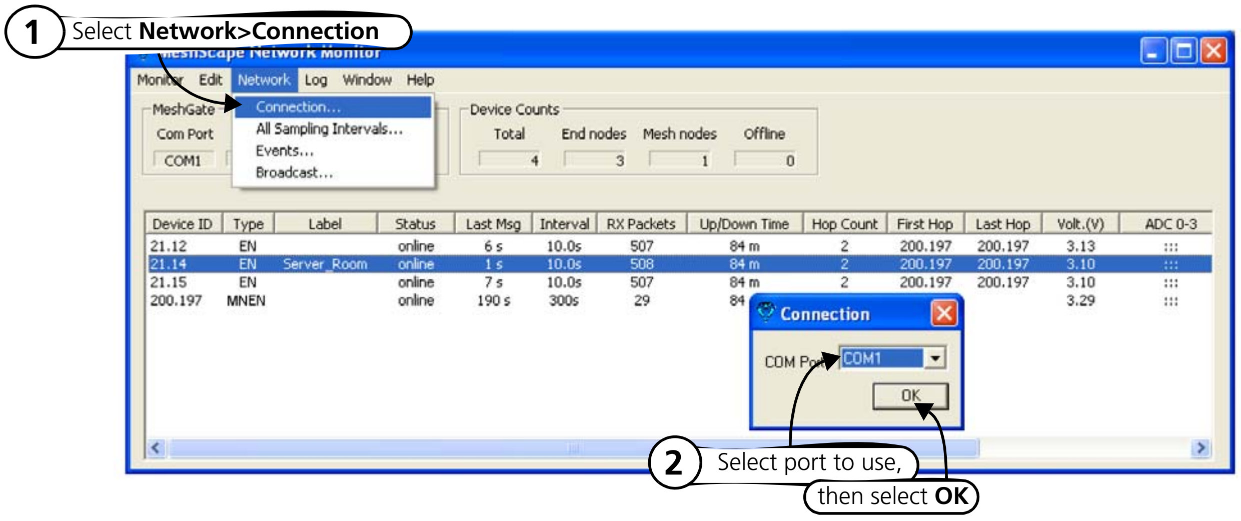
# Selecting a Com Port on the Host PC

The following procedure describes how to select the RS-232 com port for MeshScape Network Monitor to use to communicate with the MeshGate (see Figure 3-20):

1. From MeshScape Network Monitor, select **Network>Connection**. The Connection window is opened.

2. Select the RS-232 com port to use from the drop down list of available ports, then select **OK**. MeshScape Network Monitor communicates with the MeshGate connected to the selected port.

**Figure 3-20. Selecting com port on host PC**



MeshGate: Com Port COM1

Device Counts: Total 4, End nodes 3, Mesh nodes 1, Offline 0

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 6 s | 10.0s | 507 | 84 m | 2 | 200.197 | 200.197 | 3.13 | ::: |
| 21.14 | EN | Server_Room | online | 1 s | 10.0s | 508 | 84 m | 2 | 200.197 | 200.197 | 3.10 | ::: |
| 21.15 | EN | | online | 7 s | 10.0s | 507 | 84 m | 2 | 200.197 | 200.197 | 3.10 | ::: |
| 200.197 | MNEN | | online | 190 s | 300s | 29 | 84 | | | | 3.29 | ::: |

COM Port: COM1

**Note:** If the serial link to the MeshGate from the host PC is lost and then re-established, you must reset the MeshScape Network Monitor com port connection to the MeshGate using the Connection dialog, or alternatively, you can exit and then restart the MeshScape Network Monitor application.

# Configuring Serial and ADC Data Formats

The following procedure describes how to configure the format of displayed serial and ADC information (see Figure 3-21):

1. From MeshScape Network Monitor, select **Edit**>**Data format**. The Edit Data Format window is opened.

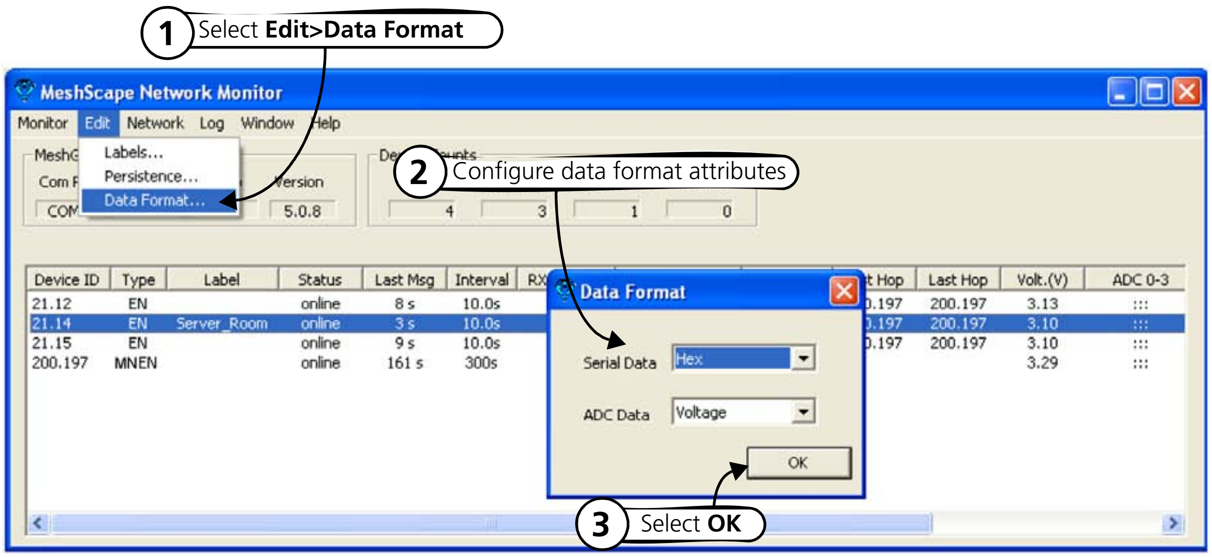
2. Configure the following network attributes:

   – **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 3-21. Configuring serial and ADC data formats**



| Device ID | Type | Label | Status | Last Msg | Interval | RX | t Hop | Last Hop | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 8 s | 10.0s | | .197 | 200.197 | 3.13 | ::: |
| 21.14 | EN | Server_Room | online | 3 s | 10.0s | | .197 | 200.197 | 3.10 | ::: |
| 21.15 | EN | | online | 9 s | 10.0s | | .197 | 200.197 | 3.10 | ::: |
| 200.197 | MNEN | | online | 161 s | 300s | | | | 3.29 | ::: |

Serial Data: Hex

ADC Data: Voltage

## Turning Event Tracking On/Off

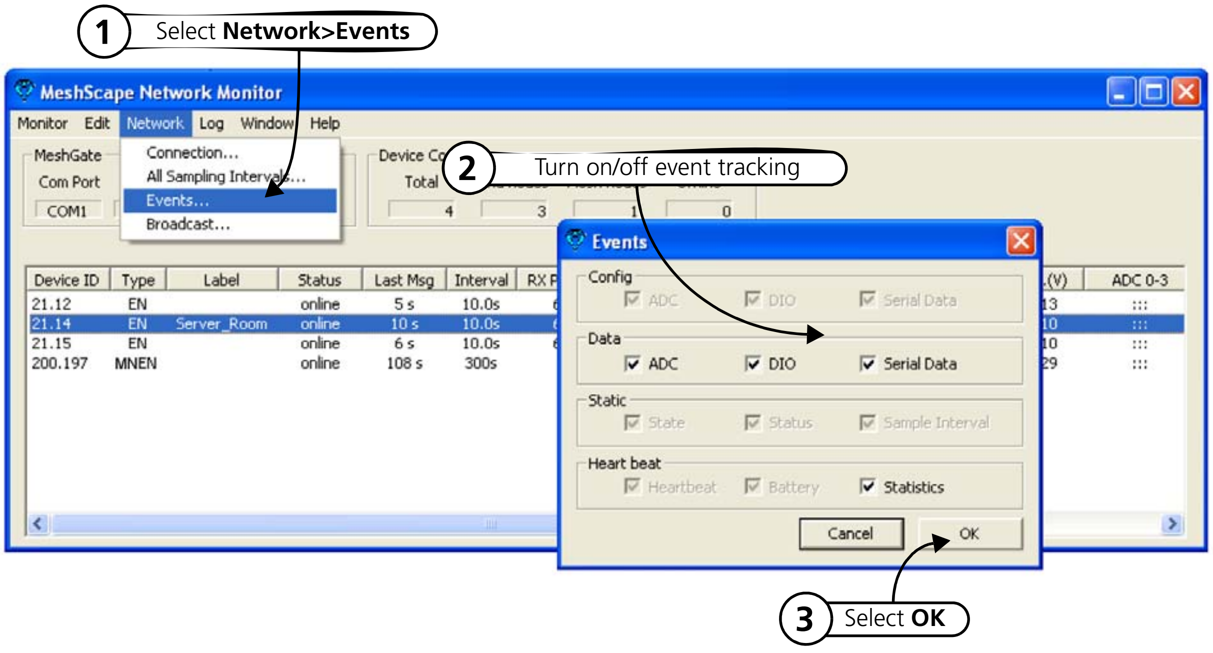
The following procedure describes how to turn event tracking on or off. Tracked events are recorded in the Event log file (see Figure 3-22):

1. From MeshScape Network Monitor, select **Network**>**Events**. The Events window is opened.
2. Configure the following event tracking settings:
   – **Configuration Events**: Display event tracking status for Analog-to-digital Converter, Digital Input/Output, and Serial configuration events.
   – **Data Events**: Turn event tracking on/off for Analog-to-digital Converter, Digital Input/Output, and Serial data events.
   – **Static Events:** Display event tracking status for device State, Status, and Sampling Interval events.
   – **Heartbeat:** Display event tracking status for device Heartbeat and Battery Level events. Display event tracking status for Statistical events.

***Note:*** Events for which tracking cannot be turned off appear grayed out in the Events window. These events are critical to proper functioning of the Network Monitor application.

3. Select **OK** to save the settings and exit the Events window.

**Figure 3-22. Turning device event tracking on/off**


1 Select **Network>Events**
MeshScape Network Monitor
Monitor Edit Network Log Window Help
Connection...
All Sampling Intervals...
Events...
Broadcast...
2 Turn on/off event tracking
Events
Cancel
OK
3 Select **OK**

MeshGate: Com Port COM1

Device Count: Total 4 | 3 | 1 | 0

| Device ID | Type | Label | Status | Last Msg | Interval |
|---|---|---|---|---|---|
| 21.12 | EN | | online | 5 s | 10.0s |
| 21.14 | EN | Server_Room | online | 10 s | 10.0s |
| 21.15 | EN | | online | 6 s | 10.0s |
| 200.197 | MNEN | | online | 108 s | 300s |

Config: ☑ ADC ☑ DIO ☑ Serial Data

Data: ☑ ADC ☑ DIO ☑ Serial Data

Static: ☑ State ☑ Status ☑ Sample Interval

Heart beat: ☑ Heartbeat ☑ Battery ☑ Statistics

## Broadcasting Data to All Nodes.

***Note:*** The broadcast feature is not supported in this release of the RK-5424-5 Reference Kit but will be supported in future releases.

You can broadcast the following data to all nodes on your MeshScape system:

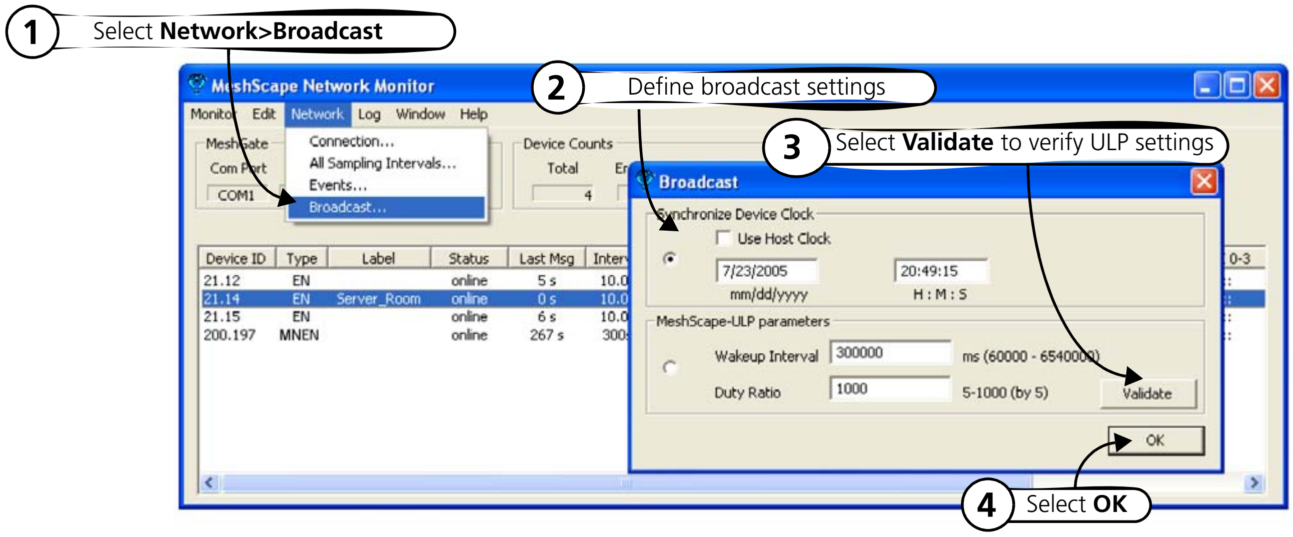
- The time to which all nodes will synchronize their clocks
- Ultra Low Power (ULP) settings defining wakeup interval and duty cycle for all nodes

The following procedure describes how to broadcast data to all nodes (see Figure 3-22):

1. From MeshScape Network Monitor, select **Network**>**Broadcast**. The Broadcast window is opened.
2. Configure the following broadcast settings:
   - **Synchronize device clock**: Set the time with which the node clocks are to synchronize. Mark the Use host clock check box to broadcast the time set on the host PC on which the MeshScape Network Monitor is running, or alternatively, enter the date and time to broadcast.
   - **Wakeup interval:** The wakeup interval determines the Mesh Node wakeup cycle. Minimum value for the wakeup interval is 60000 milliseconds, and the maximum value is 6540000 milliseconds.
   - **Duty ratio:** The duty ratio determines how long the Mesh Node stays awake and how long it sleeps during each wakeup interval. The duty ratio is defined as 100*(awake time)/(wakeup interval). Note that the wakeup interval = awake time + sleep time. The duty ratio has no unit.

     The units for specifying the duty cycle is tenths of a percent. The recommended Minimum value for the duty ratio is 5 (represent 0.5%), and the Maximum value is 1000 (represents 100.0%) set in increments of 5.
3. Select **Validate** to verify your ULP settings are valid and within range.
4. Select **OK** to save the settings and exit the Broadcast window.

**Figure 3-23. Broadcasting data to all nodes**


1 Select **Network>Broadcast**
2 Define broadcast settings
3 Select **Validate** to verify ULP settings
4 Select **OK**
MeshScape Network Monitor
Monitor Edit Network Log Window Help
Connection...
All Sampling Intervals...
Events...
Broadcast...
Broadcast
Validate
OK

MeshGate Com Port: COM1

Device Counts Total: 4

| Device ID | Type | Label | Status | Last Msg | Inter |
|---|---|---|---|---|---|
| 21.12 | EN | | online | 5 s | 10.0 |
| 21.14 | EN | Server_Room | online | 0 s | 10.0 |
| 21.15 | EN | | online | 6 s | 10.0 |
| 200.197 | MNEN | | online | 267 s | 300 |

Synchronize Device Clock (●)
- ☐ Use Host Clock
- 7/23/2005 (mm/dd/yyyy)
- 20:49:15 (H : M : S)

MeshScape-ULP parameters (○)
- Wakeup Interval: 300000 ms (60000 - 6540000)
- Duty Ratio: 1000 5-1000 (by 5)

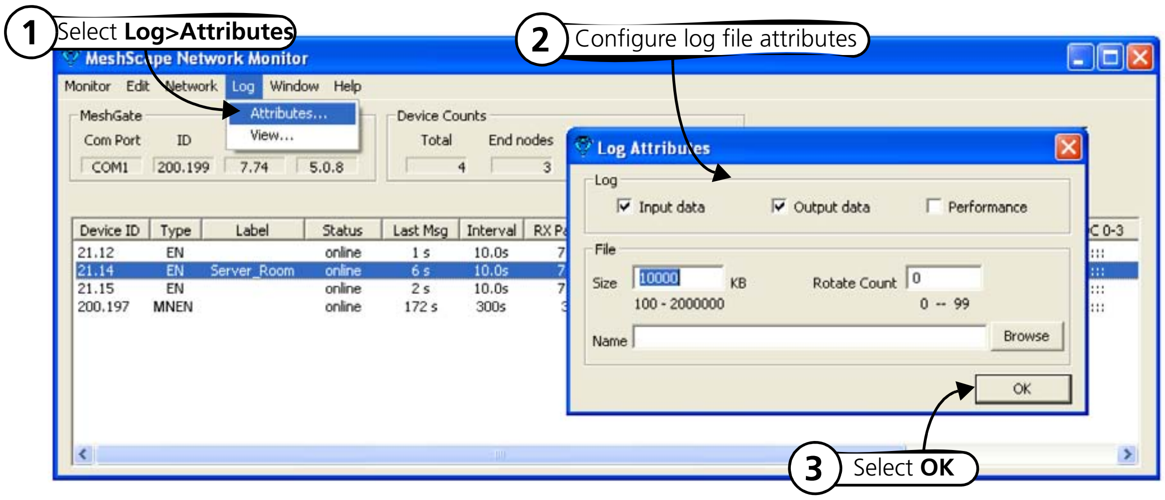
# Creating an Event Log File

The following procedure describes the steps required to have MeshScape Network Monitor record reported network events to a log file (see Figure 3-24):

1. From MeshScape Network Monitor, select **Log**>**Attributes**. The Edit Logging Attributes window is opened.
2. Configure the following Mesh Node and End Node log file attributes:
   - **Log data**: Record input/output/performance data.
   - **File size *n* KB**: Clear the log file and begin the recording process again when it reaches the designated file size and rotate count number.
   - **Rotate Count *n***: Number of history log files to maintain (not including active log file). Once this value is reached, files are written over.
   - **Name**: Assign a name to the log file and define where the file is saved.If the field is left blank or filled with an invalid pathname, logging is disabled.
3. Select **OK** to activate the logging process.

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

**Figure 3-24. Configure an event log file**


1 Select Log>Attributes
2 Configure log file attributes
3 Select OK
MeshScape Network Monitor
Monitor Edit Network Log Window Help
Attributes...
View...
Log Attributes
OK
Browse

MeshGate

| Com Port | ID | | |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5.0.8 |

Device Counts

| Total | End nodes |
|---|---|
| 4 | 3 |

| Device ID | Type | Label | Status | Last Msg | Interval | RX P |
|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 1 s | 10.0s | 7 |
| 21.14 | EN | Server_Room | online | 6 s | 10.0s | 7 |
| 21.15 | EN | | online | 2 s | 10.0s | 7 |
| 200.197 | MNEN | | online | 172 s | 300s | 3 |

Log: [✓] Input data [✓] Output data [ ] Performance

File: Size 10000 KB (100 - 2000000) Rotate Count 0 (0 -- 99)

Name

# Viewing the Contents of an Event Log File

The following procedure describes how to view the network event log file generated by MeshScape Network Monitor, which records node-generated events (see Figure 3-25):

1. From MeshScape Network Monitor, select **Log>View**. The log file window is opened, displaying the user-defined event log information (see also 'Creating an Event Log File' on page 3-25).
2. Select one of the following:
   - **Stop/Restart**: Select this toggle option to stop and restart display of the log file.
   - **Clear**: Select this option to clear the information currently displayed and start a new recording session. Clearing the window will not erase a log file from the hard disk.

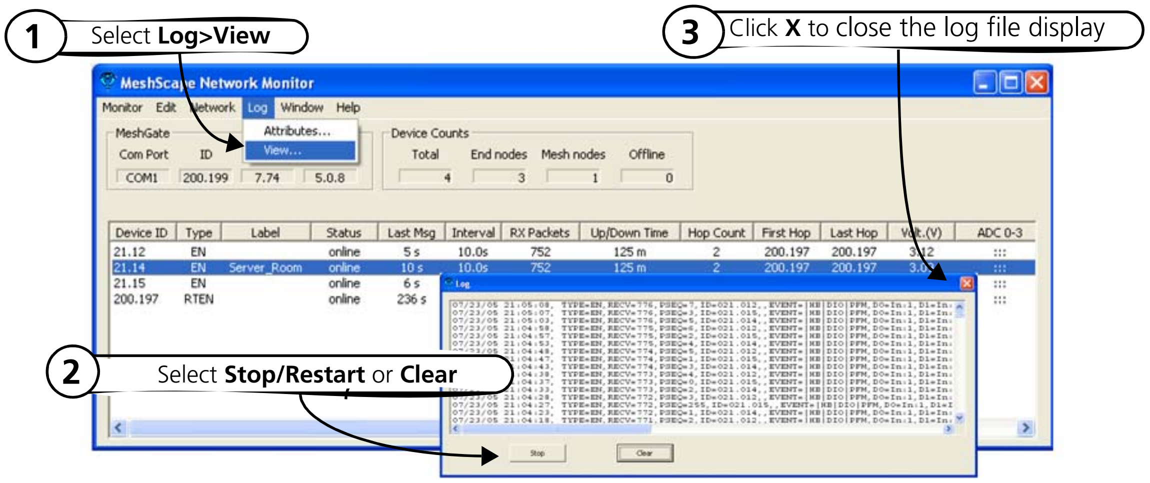
The log file window displays 2056 lines of log file data.

You can copy and paste event log data to the Windows clipboard.

To configure the events to track and include in the log file, see 'Turning Event Tracking On/Off' on page 3-22.

To configure the attributes of the log file that get recorded and displayed, see 'Creating an Event Log File' on page 3-25.

**Figure 3-25. View contents of event log file**


1 Select **Log>View**
3 Click **X** to close the log file display
2 Select **Stop/Restart** or **Clear**
MeshScape Network Monitor
Monitor
Edit
Network
Log
Window
Help
Attributes...
View...
Log
Stop
Clear

MeshGate

| Com Port | ID | | |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5.0.8 |

Device Counts

| Total | End nodes | Mesh nodes | Offline |
|---|---|---|---|
| 4 | 3 | 1 | 0 |

| Device ID | Type | Label | Status | Last Msg | Interval | RX Packets | Up/Down Time | Hop Count | First Hop | Last Hop | Volt.(V) | ADC 0-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21.12 | EN | | online | 5 s | 10.0s | 752 | 125 m | 2 | 200.197 | 200.197 | 3.12 | ::: |
| 21.14 | EN | Server_Room | online | 10 s | 10.0s | 752 | 125 m | 2 | 200.197 | 200.197 | 3.0 | ::: |
| 21.15 | EN | | online | 6 s | | | | | | | | ::: |
| 200.197 | RTEN | | online | 236 s | | | | | | | | ::: |

```
07/23/05 21:05:08,  TYPE=EN,RECV=776,PSEQ=7,ID=021.012,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:05:07,  TYPE=EN,RECV=776,PSEQ=3,ID=021.015,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:05:03,  TYPE=EN,RECV=776,PSEQ=5,ID=021.014,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:04:58,  TYPE=EN,RECV=775,PSEQ=6,ID=021.012,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:04:57,  TYPE=EN,RECV=775,PSEQ=2,ID=021.015,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:04:53,  TYPE=EN,RECV=775,PSEQ=4,ID=021.014,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:04:48,  TYPE=EN,RECV=774,PSEQ=5,ID=021.012,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
         21:04:47,  TYPE=EN,RECV=774,PSEQ=1,ID=021.015,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
           :04:43,  TYPE=EN,RECV=774,PSEQ=3,ID=021.014,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
           :04:38,  TYPE=EN,RECV=773,PSEQ=4,ID=021.012,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
           :04:37,  TYPE=EN,RECV=773,PSEQ=0,ID=021.015,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
         21:04:33,  TYPE=EN,RECV=773,PSEQ=2,ID=021.014,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:04:28,  TYPE=EN,RECV=772,PSEQ=3,ID=021.012,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:04:27,  TYPE=EN,RECV=772,PSEQ=255,ID=021.015,,EVENT=|HB|DIO|PFM,D0=In:1,D1=I
07/23/05 21:04:23,  TYPE=EN,RECV=772,PSEQ=1,ID=021.014,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
07/23/05 21:04:18,  TYPE=EN,RECV=771,PSEQ=2,ID=021.012,,EVENT=|HB|DIO|PFM,D0=In:1,D1=In:
```

Table 3-10 describes the event keys (see example) displayed in the log file.

*example*: `04/22 11:00:42, Type=EN, RCV=100, PSEQ=28, ID=021.015,`
`Event=|HB|DIO|PFM, D0=In:1, D1=In:1, D2=In:1, D3=In:1`

**Table 3-10.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 | |
| EP | Endpoint | N/A | N/A | EP=001.001 | EP=001.002 |
| FH | First Hop Mesh Node | N/A | N/A | N/A | FH=201.002 |
| HC | Hop Count | N/A | N/A | N/A | HC=3 |
| LH | Last Hop Mesh Node | N/A | N/A | N/A | LH=202.003 |
| RCV | Receive | N/A | N/A | N/A | RCV=200 (sequence number) |
| MNEN | Mesh Node | 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 | | |
| | | Ref | Refreshing | | |

## Viewing MeshScape Statistics

The following procedure describes how to view the network statistics recorded by MeshScape Network Monitor, which provides information on the packets received and transmitted by the network nodes (see Figure 3-26):

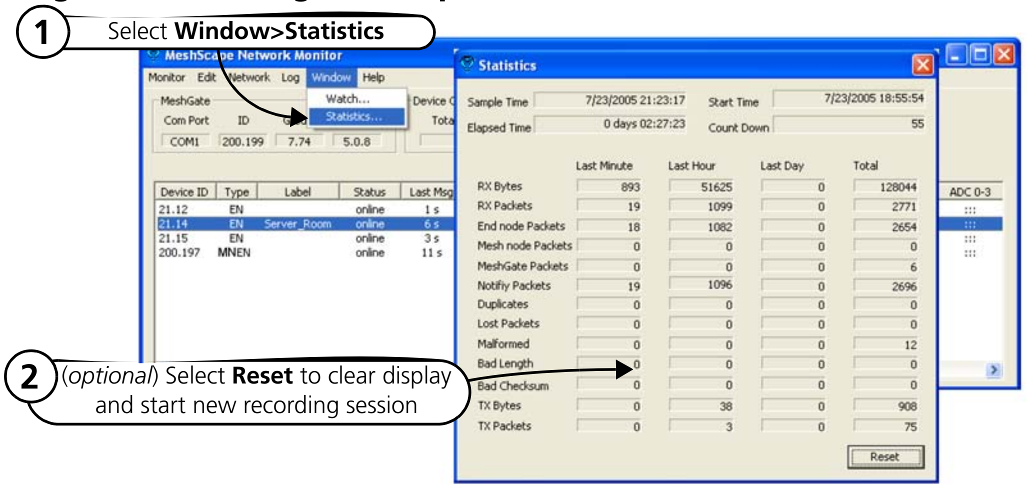
1. From MeshScape Network Monitor, select **Window**>**Statistics**. The Statistics window is opened, displaying the following information:
   - **Sample Time:** Current date and time.
   - **Start Time**: Date/time statistics recording session began.
   - **Elapsed Time**: Total recording time.
   - **Count Down:** Seconds before next sample is taken and window is refreshed.
   - **RX Bytes**: Total bytes received.
   - **RX Packets**: Total packets received.
   - **End Node Packets**: Packets received from End Nodes.
   - **Mesh Node Packets**: Packets received from Mesh Nodes.
   - **MeshGate Packets**: Packets received from MeshGate.
   - **Notify Packets**: Packets sent from MeshGate to MeshScape Network Monitor for notification events.
   - **TX Bytes**: Total bytes transmitted.
   - **TX Packets**: Total packets transmitted.

**Caution**

Selecting **Reset** will remove all statistics collection history.

2. (*optional*) Select **Reset** to clear the information currently displayed and begin a new recording session.

**Figure 3-26. Viewing MeshScape statistics**


1
Select **Window>Statistics**
2
(*optional*) Select **Reset** to clear display and start new recording session
MeshScape Network Monitor
Monitor
Edit
Network
Log
Window
Help
Watch...
Statistics...
Statistics
Reset

| MeshGate Com Port | ID | | |
|---|---|---|---|
| COM1 | 200.199 | 7.74 | 5.0.8 |

| Device ID | Type | Label | Status | Last Msg |
|---|---|---|---|---|
| 21.12 | EN | | online | 1 s |
| 21.14 | EN | Server_Room | online | 6 s |
| 21.15 | EN | | online | 3 s |
| 200.197 | MNEN | | online | 11 s |

| | | | |
|---|---|---|---|
| Sample Time | 7/23/2005 21:23:17 | Start Time | 7/23/2005 18:55:54 |
| Elapsed Time | 0 days 02:27:23 | Count Down | 55 |

| | Last Minute | Last Hour | Last Day | Total |
|---|---|---|---|---|
| RX Bytes | 893 | 51625 | 0 | 128044 |
| RX Packets | 19 | 1099 | 0 | 2771 |
| End node Packets | 18 | 1082 | 0 | 2654 |
| Mesh node Packets | 0 | 0 | 0 | 0 |
| MeshGate Packets | 0 | 0 | 0 | 6 |
| Notifiy Packets | 19 | 1096 | 0 | 2696 |
| Duplicates | 0 | 0 | 0 | 0 |
| Lost Packets | 0 | 0 | 0 | 0 |
| Malformed | 0 | 0 | 0 | 12 |
| Bad Length | 0 | 0 | 0 | 0 |
| Bad Checksum | 0 | 0 | 0 | 0 |
| TX Bytes | 0 | 38 | 0 | 908 |
| TX Packets | 0 | 3 | 0 | 75 |

# 4

# Using the MeshScape API

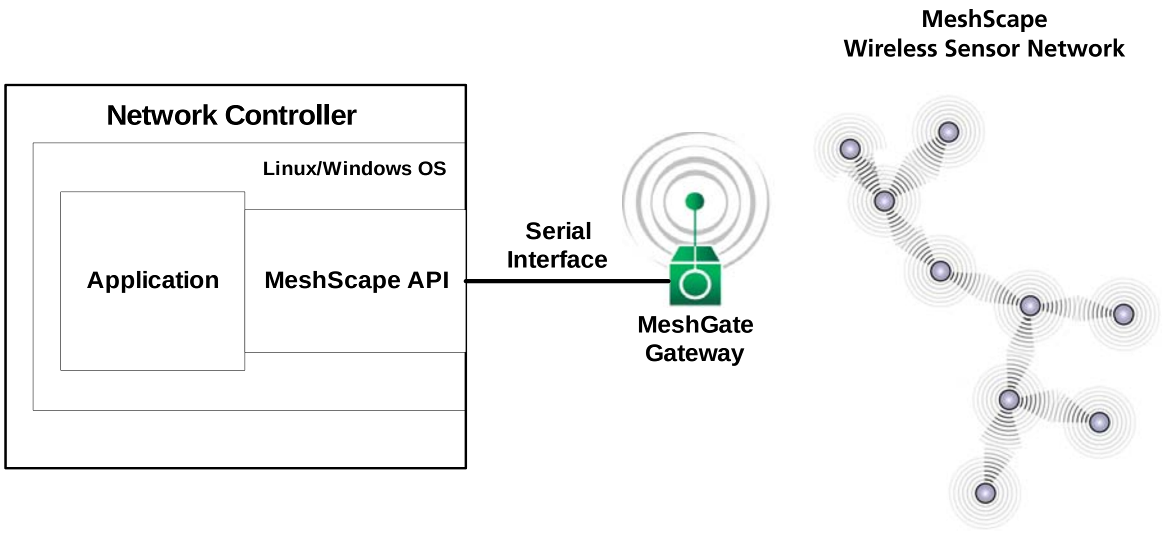
This chapter describes the following API functions:

# Using the MeshScape API

Millennial Net's MeshScape™ API enables you to effectively use the features provided by Millennial Net's MeshScape wireless sensor network. You can develop custom applications that utilize the MeshScape API library on Windows- or Linux-based platforms such as Network Controllers, PCs, or PDAs.

The Millennial API provides a serial command set that enables a MeshScape MeshGate standalone gateway to communicate with user applications via a serial interface. The serial interface transports a byte stream-based protocol that provides MeshScape API equivalent functionality. The commands operate in both a query/response and an autonomous event notification manner.

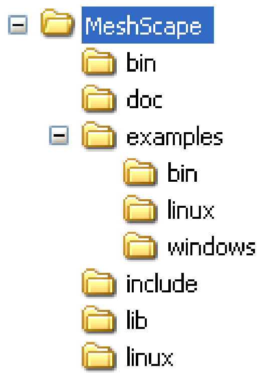
**Figure 4-1. Using the MeshScape API**



# MeshScape API Directory Structure

The MeshScape API 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 4-2 is a representation of the directory structure.

**Figure 4-2. MeshScape API directories**



The contents of each directory is described below.

**bin Directory:**

This directory contains the iBeanAPI.dll file required for running API related applications. This dll file is compiled with Microsoft's Visual Studio .Net edition and therefore only supports Microsoft Visual C++ programming conventions. The directory also contains the MeshScape Programmer (MNstall.exe) and MeshScape Network Monitor (sagMon.exe) executables.

**doc Directory:**

This directory contains the MeshScape system user documentation in Adobe Acrobat (.pdf) format including:

– RK-5424-5 Reference Kit User's Guide (i.e., this guide)
– RK-5424-5 2.4 GHz MeshScape™ Reference Kit Contents & Getting Started Guide
– MeshScape Product Family Sheet
– Technical specifications for MeshGate gateway, mesh node, and end node
– Release notes

**examples Directory:**

This directory contains three sub-directories:

– **bin** - This sub-directory contains the demo.exe and application.dll files required to run the demonstration applications expressly developed to demonstrate the capabilities of MeshScape. Running the temperature sensor demonstration application is described in Appendix A. The Model3_temp.txt file in the directory is required for the temperature sensor demo and should not be moved or deleted.

- **linux** - The source code for sample applications to be compiled under Linux:

  ListDevices.c: This is a console based application that lists current online devices. For a detailed look at this example and its code, see 'Example API Code' on page 4-45.

  ReadSerial.c: This is a console based application that reads serial data received from any online device on the network that has serial interface enabled.

  SetSampling.c: This is a console based application that changes the sampling interval of the online devices.

- **windows** - The source code for sample applications to be compiled under Windows.

  ListDevices.c: This is a console based application that lists current online devices. For a detailed look at this example and its code, see 'Example API Code' on page 4-45.

  ReadSerial.c: This is a console based application that reads serial data received from any online device on the network that has serial interface enabled.

  SetSampling.c: This is a console based application that changes the sampling interval of the online devices.

### include Directory:

This directory contains the MeshScape API header files required to build any API-based application under Windows. These header files are documented in this chapter.

### lib Directory:

This directory contains the iBeanAPI.lib file required to compile any API-based application under Windows.

### linux Directory:

This directory contains a single compressed tar file that provides the MeshScape API header files and shared object files required to build an API-based application under Linux.

# MeshScape API Functions Overview

Table 4-1 provides a list of API functions associated with MeshScape system products.

**Table 4-1. MeshScape API functions**

| iBeanAPI.h | Core API functions |
|---|---|
| ibApi_Open()<br>ibApi_Close()<br>ibApi_GetApiVersion() | Session Management |
| ibApi_GetNetworkList()<br>ibApi_GetDeviceList() | Enumeration of network devices |
| ibApi_GetDeviceInfo()<br>ibApi_GetDeviceStatus() | Static and dynamic device attributes |
| iBApi_GetDeviceState()<br>ibApi_SetSamplingInterval()<br>ibApi_GetSamplingInterval() | Universally supported device properties |
| ibApi_WaitForDeviceEvent()<br>ibApi_WaitForDeviceEvents()<br>ibApi_SetEventMask()<br>ibApi_GetEventMask()<br>ibApi_GetDevicePacketSequenceNumber()<br>ibApi_Block()<br>ibApi_UnBlock()<br>ibApi_RegisterEvent() | Event Notification |
| iBeanAPI_IO.h | Standard I/O peripherals |
| ibApi_IO_GetDeviceCaps() | Static device attributes |
| ibApi_IO_SetADCConfig()<br>ibApi_IO_GetADCConfig()<br>ibApi_IO_ReadADC()<br>ibApi_IO_ReadRawADC() | Analog-to-Digital conversion |
| ibApi_IO_SetDIOConfig()<br>ibApi_IO_GetDIOConfig()<br>ibApi_IO_WriteDIO()<br>ibApi_IO_ReadDIO() | Digital input/output |

**Table 4-1. MeshScape API functions (continued)**

| | |
|---|---|
| ibApi_IO_SetSerialConfig()<br>ibApi_IO_GetSerialConfig()<br>ibApi_IO_GetSerialBufferStatus()<br>ibApi_IO_WriteSerial()<br>ibApi_IO_ReadSerial()<br>ibApi_IO_SetDeviceConfigAndData() | Serial data interface (UART) |
| iBeanAPI_Utils.h | Supplementary functions |
| ibApi_Utils_GetErrorDescription() | Obtain text descriptions for error codes |
| ibApi_Utils_ConvertGroupIdToText()<br>ibApi_Utils_ConvertTextToGroupId()<br>ibApi_Utils_ConvertDeviceIdToText()<br>ibApi_Utils_ConvertTextToDeviceId()<br>ibApi_Utils_ConvertRawBatteryReadingToVoltage()<br>ibApi_Utils_ConvertRawAdcReadingToVoltage() | Convert between ID structures and text representation |
| iBeanAPI_LPR.h | Ultra-Low Power Functions |
| ibApi_LPR_SetClock() | Set the Coordinated Universal Time (UTC) clock for all devices |
| ibApi_LPR_GetClock() | Retrieves the UTC clock value used for all devices |
| ibApi_LPR_SetWakeupAndDutyRatio() | Sets the low power mesh node wakeup interval and duty ratio for all devices |
| ibApi_LPR_GetWakeupAndDutyRatio() | Retrieves the wakeup interval and duty ratio for all devices |
| iBeanAPI_performance.h | Performance Statistics |
| ibApi_GetStatisticData() | Retrieve statistical data for a given device. |

# iBeanAPI.h

## Data Structures

1. **ibApi_APIHANDLE**

```
typedef ibApi_INT32 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 ibApi_INT32 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[ibApi_GROUPID_SIZE];
};
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 MeshScape system group can only have one MeshGate.)

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[ibApi_DEVICEID_SIZE];
};
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 end node, mesh node, or gateway.

5. **`ibApi_COMPARISON`**

```
enum ibApi_COMPARISON_e {
  ibApi_COMPARISON_EQUAL            = (1<<0),
  ibApi_COMPARISON_LESSTHAN         = (1<<1),
  ibApi_COMPARISON_GREATERTHAN      = (1<<2),

ibApi_COMPARISON_NOTEQUAL         =
ibApi_COMPARISON_LESSTHAN|ibApi_COMPARISON_GREATERTHAN,
  ibApi_COMPARISON_NOTLESSTHAN      =
ibApi_COMPARISON_EQUAL|ibApi_COMPARISON_GREATERTHAN,
  ibApi_COMPARISON_NOTGREATERTHAN   =
ibApi_COMPARISON_EQUAL|ibApi_COMPARISON_LESSTHAN,
  ibApi_COMPARISON_BITMASK          = 0xf
};
typedef ibApi_UINT16 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          = 0x3f  /* used with filter */
};
typedef ibApi_UINT32 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_UINT16 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 (end node, mesh node, etc.). |
| *hardware_version*<br>*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_OFFLINE,
   ibApi_DEVICESTATE_ONLINE,
   ibApi_DEVICESTATE_REFRESHING,
};
typedef ibApi_UINT16 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_UINT16 struct_size;
   ibApi_UINT16 hop_count;
   ibApi_DEVICEID first_hop_router;
   ibApi_DEVICEID last_hop_router;
   ibApi_FLOAT battery_level;
   ibApi_UINT16 battery_level_raw
   ibApi_UINT16 seq_num;
   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 mesh nodes), then the hop count is 1. |
| *first_hop_router* | These fields store the device ID of the first and last mesh node 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 mesh node 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 voltage.<br>If battery information is unavailable, the value is 0. |
| *battery_level_raw* | This reports the device battery level measured in raw data, not voltage. If battery information is unavailable, the value is 0.<br>Application can use ibApi_Utils_ConvertRawBatteryReadingToVoltage() to convert raw data to voltage. |
| seq_num | The sequence number increments whenever an update occurs. |
| group_id | This reports the group ID currently assigned to the device. |

11. **ibApi_EVENTMASK**

```
typedef ibApi_UINT32 ibApi_EVENTMASK;

#define ibApi_MAX_SERIAL_DATA_SIZE              255
#define ibApi_MAX_ADC_DATA_CHANNELS             4
#define ibApi_MAX_DIO_DATA_CHANNELS             4
#define ibApi_UNDEFINED_FIELD_VALUE             0xFF

#define ibApi_Serial_Receive_EVENT_DATA_TYPE    0x11
#define ibApi_ADC_EVENT_DATA_TYPE               0x22
#define ibApi_DIO_EVENT_DATA_TYPE               0x33
#define ibApi_Device_State_EVENT_DATA_TYPE      0x44
#define ibApi_Sampling_Interval_EVENT_DATA_TYPE 0x55
#define ibApi_Battery_Level_EVENT_DATA_TYPE     0x66
#define ibApi_Network_EVENT_DATA_TYPE           0x77
#define ibApi_Heartbeat_EVENT_DATA_TYPE         0x88
```

Use this structure to identify events passed to event callbacks.

12. **ibApi_CALLBACK_DATA**

```
typedef ibApi_UINT8 ibApi_CALLBACK_DATA;
```

ibApi_CALLBACK_DATA is a pointer which must be cast to one of the data types that are described in the following sections.

13. **ibApi_ADC_EVENT_DATA**

```
typedef struct {

ibApi_UINT8  dataType;         // Must be first!
ibApi_UINT8  status; // 1=enabled, 0=disabled, 0xFF=undefined
ibApi_DEVICEID    deviceID;
ibApi_UINT8       channelMask;      // 0 = no data
ibApi_UINT8       reserved;         // for alignment
ibApi_FLOAT       channelData[8];
}
```

This data structure will be passed to event callbacks to convey information about ADC events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

14. **ibApi_DIO_EVENT_DATA**

```
typedef struct {
ibApi_UINT8       dataType;         // Must be first!
ibApi_UINT8  direction; // 1=input, 0=output, 0xFF=undefined
ibApi_DEVICEID    deviceID;
ibApi_UINT8       channelMask;      // 0 = no data
ibApi_UINT8       channelData;
}
```

This data structure will be passed to event callbacks to convey information about DIO events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

15. **ibApi_Device_State_EVENT_DATA**

```
typedef struct {
ibApi_UINT8       dataType;         // Must be first!
ibApi_UINT8       reserved;         // for alignment
ibApi_DEVICEID    deviceID;
ibApi_DEVICESTATE state;
}
```

This data structure will be passed to event callbacks to convey information about device state events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

16. **ibApi_Sampling_Interval_EVENT_DATA**

```
typedef struct {
ibApi_UINT8       dataType;         // Must be first!
ibApi_UINT8       reserved;         // for alignement
ibApi_DEVICEID    deviceID;
ibApi_UINT32      samplingInterval; // in milliseconds
}
```

This data structure will be passed to event callbacks to convey information about sampling interval events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

17. **`ibApi_Serial_Receive_EVENT_DATA`**

```
typedef struct {
ibApi_UINT8       dataType;         // Must be first!
ibApi_UINT8       serialState;      // 0xFF=undefined
ibApi_DEVICEID    deviceID;
ibApi_UINT8       serialDataSize;
ibApi_UINT8       serialSequence;
ibApi_UINT8       serialData[ibApi_MAX_SERIAL_DATA_SIZE];
}
```

This data structure will be passed to event callbacks to convey information about serial receive events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

18. **`ibApi_Battery_Level_EVENT_DATA`**

```
typedef struct {
ibApi_UINT8       dataType;         // Must be first!
ibApi_UINT8       reserved;         // for alignement
ibApi_DEVICEID    deviceID;
ibApi_FLOAT       batteryLevel;
}
```

This data structure will be passed to event callbacks to convey information about battery level events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

19. **`ibApi_Network_EVENT_DATA`**

```
typedef struct {
ibApi_UINT8       dataType;         // Must be first!
ibApi_UINT8       hopCount;
ibApi_DEVICEID    deviceID;
ibApi_DEVICEID    firstHopRouter;
ibApi_DEVICEID    lastHopRouter;
}
```

This data structure will be passed to event callbacks to convey information about network events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

20. **`ibApi_Heatbeat_EVENT_DATA`**

```
typedef struct {
ibApi_UINT8       dataType;         // Must be first!
ibApi_BOOL        heartBeat;        // Will always be set to
ibApi_TRUE
ibApi_DEVICEID    deviceID;
}
```

This data structure will be passed to event callbacks to convey information about heart beat events.

Use the first argument ibApi_EVENTMASK to determine how to cast the callback data.

21. **`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)
```

These macros encode API 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.

22. **`ibApi_EXPECTED_VERSION`**

```
#define ibApi_EXPECTED_MASC_VERSION
    ibApi_MAKE_VERSION(N,N,N)
```

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. Note that N, N, N above represents the API version number, for example, 5.0.13.

## 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:** For the current release, the server_type should always be "local", and the default connection string is"". These text strings are case-sensitive.

You can change the default COM port and Baud rate using the "connection_str" parameters. The string format is "-p[n]|-B[baud rate]", where n = (COM port number – 1). For example, to select COM port 2 with a baud rate of 115200, you should supply the connection_str "-p1 –B115200". By default, the COM port is COM1 and the baud rate is 115200.

**Notes:** The connection baud rate needs to be consistent between the host and the MeshGate gateway. The default baud rate on the gateway is 115200, so if ibApi_Open() changes the baud rate to a different number, the baud rate on the gateway also needs to be updated.

To access the COM port setting on Linux platforms, you need to either have root permission or change the COM port permission to current user.

**Parameters**:

*expected_version*: (input) Should always be ibApi_EXPECTED_VERSION.

*server_type*: (input) "local" (Only valid setting for this release).

*connection_str*: (input) "" (NULL string to set to default values).

**Return Value**:

An ibApi_APIHANDLE value if successful, or an 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 function 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) Pointer to an array of device IDs.

*devices_size*: (input) Maximum number of device IDs that the devices array can hold.

**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_DEVICEINFO` above for details.

**Note**: To avoid memory corruption, the size of (`ibApi_DECVICEINFO`) must be allocated 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.

**Return Value**:

Sampling interval if successful, error code (<0) if not.

**Notes on Event Notification and Event Masks** - It is important to note that there are two techniques for receiving event notification: either by registering a callback which will be called when the event(s) occur or by waiting (blocking) until the desired event(s) occur.

When using the callback mechanism the event mask must be registered by using ibApi_SetEventMask(). This API will register a global event mask for all devices in the network. When any of the events specified in the mask occur for any device in the network, the callback registered via ibApi_RegisterEvent() will be called. If there is data associated with the event, it will be passed to the callback function.

When using the wait (blocking) mechanism an event mask is specified as an argument to the wait call (see api definitions below). NOTE that if a global event mask is in place (set with ibApi_SetEventMask()) ONLY THE EVENTS that occur in the INTERSECTION of the two event masks will be received while waiting. This allows you to specify a general event mask for your entire network, and to filter on a very specific subset of events specified in the global mask. Also note that when using the wait mechanism that an additional call is required to retreive the data (if any) associated with the event.

11. **`ibApi_WaitForDeviceEvent()`**

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

**Note:** This function is now obsolete and only retained for backward compatibility. Please use ibApi_WaitForDeviceEvents() instead.

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.

*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.

12. **`ibApi_WaitForDeviceEvents()`**

```
ibApi_FUNC ibApi_WaitForDeviceEvents(
ibApi_APIHANDLE                    api_hdl,
ibApi_DEVICEEVENTTYPE              events_interested,
ibApi_INT32                        timeout_ms,
ibApi_DEVICEID *                   device_id,
ibApi_DEVICEEVENTTYPE *            events_happened
);
```

This function implements the simplest form of event notification using the application thread context. 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 along with the event(s) that woke the sleeping thread up. If multiple devices have changed since the last call, ibApi_WaitForDeviceEvent() will return their IDs in a sequential "round robin" order. If time timeout expires and nothing has changed, the return value is ibApi_RESULT_ERR_TIMEOUT.

**Parameters**:

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

*events_interested*: (input) This parameter allows the wait condition to be restricted to a subset of the possible event types.

*timeout_ms*: Number of milliseconds to wait before giving up (use -1 to wait indefinitely) A valid of zero means the thread will check for new events since the last call but will not wait.

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

*events_happened:* (output) The returned value is the subset of the events_interested. It indicates the event(s) that actually happened.

**Return Value**:

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

## 13. **ibApi_SetEventMask()**

```
ibApi_SetEventMask(
ibApi_APIHANDLE        api_hdl,
ibApi_DEVICEEVENTTYPE  event,
ibApi_DEVICEID         device_id
);
```

This function sets the event mask for the triggering of asynchronous events. Before calling ibApi_SetEventMask(), you must set the device_id to 0.0.

**Parameters**:

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

*event*: (input) The event mask.

*device_id*: (input) ID of the device that changed.

**Return Value**:

An ibApi_RESULT_SUCCESS if a device changed or an error code (<0) if unsuccessful.

## 14. **ibApi_GetEventMask()**

```
ibApi_FUNC ibApi_GetEventMask(
ibApi_APIHANDLE        api_hdl,
ibApi_DEVICEEVENTTYPE *event,
ibApi_DEVICEID         device_id
);
```

This function gets the event mask for the triggering of asynchronous events.

**Parameters**:

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

*event*: (output) The event mask storage location.

*device_id*: (input) ID of the device that changed.

**Return Value**:

An ibApi_RESULT_SUCCESS if a device changed or an error code (<0) if unsuccessful.

## 15. **ibApi_RegisterEvent()**

```
ibApi_RegisterEvent(
ibApi_APIHANDLE          api_hdl,
ibApi_CALLBACK_FUNCTIONapi_callback
);
```

Registers a callback function to be called upon receipt of the specified asynchronous event.

See “Notes on Event Notification abd Event Masks” on page 4-20.

A pointer to a function which implements this type must be passed to ibApi_RegisterEvent( ).

Each event processed will result in a call to this callback. The event mask will contain a single event indicator (ibApi_DEVICEEVENTTYPE). Use the indicator to determine how to cast ibApi_CALLBACK_DATA*.

@param ibApi_EVENTMASK (input) event mask, set to only one event type

@param ibApi_CALLBACK_DATA* (input) event data

typedef void (*ibApi_CALLBACK_FUNCTION)(ibApi_EVENTMASK, ibApi_CALLBACK_DATA *);

**Parameters**:

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

*api_callback*: (input) Callback function to register for asynchronous events as defined by prior call to ibApi_SetEventMask().

**Return Value**:

An ibApi_RESULT_SUCCESS if successful or an error code (<0) if unsuccessful.

## 16. **ibApi_GetDevicePacketSequenceNumber()**

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

This function Gets a device packet sequence number that is carried by a upstream packet sent from the device.

**Parameters**:

| | |
|---|---|
| *api_hdl:* | (input) API handle returned from ibApi_Open(). |
| *device_id:* | (input) The device id. |

**Return Value**:

A 0 - 255 sequence number if successful, or an error code (<0) if failed.

17. **`ibApi_Block (void)`**

```
ibApi_FUNC ibApi_Block(void);
```

This function provides locking of shared data areas from within the application event callback. This function is to be called prior to accessing the data record returned to the callback function.

**Parameters**:

*None.*

**Return Value**:

An ibApi_RESULT_SUCCESS if a device changed or an error code (<0) if unsuccessful.

18. **`ibApi_UnBlock (void)`**

```
ibApi_FUNC ibApi_UnBlock(void);
```

This function releases lock of shared data area from within the application event callback function. This must be called prior to exiting the application event callback if the ibApi_Block function was previously invoked.

**Parameters**:

*None.*

**Return Value**:

An ibApi_RESULT_SUCCESS if a device changed 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 = 0,
  ibApi_IO_SERIALMODE_UART,
  ibApi_IO_SERIALMODE_RS232,
  ibApi_IO_SERIALMODE_RS485
};
typedef ibApi_UINT16 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_UINT16 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_UINT16 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 as a float voltage value.

**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) ADC channel voltage.

**Return Value**:

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

## 5. **ibApi_IO_ReadRawADC()**

```
ibApi_FUNC ibApi_IO_ReadRawADC(
   ibApi_APIHANDLE     api_hdl,
   ibApi_DEVICEID      device_id,
   ibApi_UINT8         channel_index,
   ibApi_UINT16*       adc_value
);
```

This retrieves the value of the specified ADC channel in raw format. Use this call only if a raw value is explicitly required, otherwise use ibApi_IO_ReadADC() which returns a float value voltage. The raw return value may be converted to a voltage using ibApi_Utils_ConvertRawAdcReadingToVoltage() found in iBeanAPI_Utils.h.

**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 raw data, not voltage. Application can use ibApi_Utils_ConvertRawBatteryReadingToVoltage() to convert to voltage.

**Return Value**:

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

## 6. **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) Input/output mode.

**Return Value**:

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

## 7. **ibApi_IO_GetDIOConfig()**

```
ibApi_FUNC ibApi_IO_SetDIOConfig(
  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, 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) the new input/output mode

**Return Value**:

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

## 8. **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.

9. **`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).

10. **`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.

## 11. **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.

## 12. **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 number of empty out going packet slots if successful, or an error code (<0) if not.

13. **`ibApi_IO_WriteSerial()`**

```
ibApi_FUNC ibApi_IO_WriteSerial (
   ibApi_APIHANDLE     api_hdl,
   ibApi_DEVICEID      device_id,
   ibApi_UINT8         buffer[],
   ibApi_UINT16        buffer_size
);
```

This function writes buffer_size bytes pointed to by the buffer pointer to the specified device handle. Prior to use, the ibApi_FIELDID_USERDATAMODE field must have been configured for serial operation. The specific contents of the user data block and its maximum size are application defined but must be equal to or smaller than that the maximum payload size supported.  Maximum payload size supported is returned when the function `ibApi_WriteSerialData()` is called 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) Pointer to packet to transmit.

*buffer_size*: (input) Number of bytes in user data packet to transmit.

**Return Value**:

Bytes sent if successful, or an error code (<0) if not.

## 14. **ibApi_IO_ReadSerial()**

```
ibApi_FUNC ibApi_IO_ReadSerial (
   ibApi_APIHANDLE     api_hdl,
   ibApi_DEVICEID      device_id,
   ibApi_UINT8         buffer[],
   ibApi_UINT16        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) Pointer to 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)

## 15. **ibApi_IO_SetDeviceConfigAndData()**

```
ibApi_FUNC ibApi_IO_SetDeviceConfigAndData(
  ibApi_APIHANDLE api_hdl,
  ibApi_DEVICEID  device_id,
  ibApi_deviceSetConfigAndData *config_data
);
```

This function is used to make several configuration and data requests to a device at one time.

**Parameters**:

*api_hdl:* (input) API handle returned from @link ibApi_Open()

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

*config_data*: (input) User requested configuration.

# 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_GROUPIDgroup_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 “1.2.3.4” would be rendered as “2.3.4”.

**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 "1.2.3.4.5.6.7.8" would be rendered as "5.6.7.8".

**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 function 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**:

*device_id_text*: (input) Buffer to be parsed.

*device_id*: (output) Structure to store the result.

**Return Value**:

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

6. **`ibApi_Utils_ConvertRawBatteryReadingToVoltage()`**

```
ibApi_FUNC ibApi_Utils_ConvertRawBatteryReadingToVoltage(
   ibApi_DEVICEID device_id,
   ibApi_UINT16 blRaw,
   ibApi_FLOAT *blFloat
);
```

This function converts a device’s battery level reading from raw data to voltage.

**Parameters**:

*device_id:* (input) Device information.

*blRaw*: (input) Battery level reading in raw data.

*blFloat:* (output) Structure to store the battery level reading as a voltage.

**Return Value**:

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

7. **`ibApi_Utils_ConvertRawAdcReadingToVoltage()`**

```
ibApi_FUNC ibApi_Utils_ConvertAdcReadingToVoltage(
   ibApi_DEVICEID device_id,
   ibApi_UINT16 adcRaw,
   ibApi_FLOAT *adcFloat
);
```

This function converts a device’s battery level reading from raw data to voltage.

**Parameters**:

*device_id:* (input) Device information.

*adcRaw*: (input) ADC reading in raw data.

*adcFloat:* (output) Structure to store the ADC reading as a voltage.

**Return Value**:

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

# iBeanAPI_LPR.h

## Functions

1. **`ibApi_LPR_SetClock()`**

```
ibApi_FUNC ibApi_LPR_SetClock(
   ibApi_APIHANDLE     api_hdl,
   ibApi_GROUPID       network,
   ibApi_UINT32        utc_sec
);
```

This function sets the UTC clock for all devices. The UTC clock is used for the time synchronization purposes. The gateway will synchronize each device's clock through time synchronization process. The clock setting is a global attribute that applies to all devices in the network.

**Parameters**:

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

*network*: (input) Group ID of the network.

*utc_sec*: (input) The UTC value in seconds, system clock is used if the value is zero.

**Return Value**:

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

2. **`ibApi_LPR_GetClock()`**

```
ibApi_FUNC ibApi_LPR_GetClock(
   ibApi_APIHANDLE     api_hdl,
   ibApi_GROUPID       network,
   ibApi_UINT32 *      utc_sec
)
```

This function retrieves the UTC clock value (in seconds) used for all devices.

**Note**: After setting the network-wide clock by the method ibApi_LPR_SetClock(), it is necessary to wait at least 1 second before calling this function to allow the system to take on the new value.

**Parameters**:

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

*network*: (input) Group ID of the network.

*utc_sec*: (output) UTC value in seconds.

**Return Value**:

The wakeup interval in ms if successful, or error code (<0) if not.

3. **`ibApi_LPR_SetWakeupAndDutyRatio()`**

```
ibApi_FUNC ibApi_LPR_SetWakeupAndDutyRatio(
   ibApi_APIHANDLE     api_hdl,
   ibApi_GROUPID       network,
   ibApi_UINT32        wakeup_interval,
   ibApi_UINT32        duty_ratio
)
```

This function sets the low power router wakeup interval and duty ratio for all devices. The wakeup interval and duty ratio are global attributes that applies to all devices in the network.

Changing of wakeup interval and duty ratio may have impact on the network stability, however, the worst case is to cause the network to reestablish itself entirely. Minimum value for wakeup interval is 60 seconds, maximum value is 6540 seconds (109 minutes).

Wakeup interval should be expressed in milliseconds. If too many digits of precision are used round off error can occur. Use ranges of 100's, 1000's, 10000's or 100000's of milliseconds rounded up to the nearest second to prevent round off error.

Minimum value for duty radio depends on minimum wakeup interval, and network size, radius (number of hops), currently it set to be 10%, maximum value is 100%. The 100% duty ratio means the mesh node is no longer asleep.

Duty ratio can be set in 0.5% increments using the integer range 5-1000, with increments of 5. If (DutyRatio MOD 5) != 0, an error will be returned.

**Parameters**:

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

*network*: (input) Group ID of the network.

*wakeup_interval*: (input) Wakeup interval in milliseconds.

*duty_ratio*: (input) the duty ratio (5 - 1000), in increments of 5.

**Return Value**:

Wakeup interval in ms if successful, error code (<0) if not.

4. **`ibApi_LPR_GetWakeupAndDutyRatio)`**

```
ibApi_FUNC ibApi_LPR_GetWakeupAndDutyRatio(
   ibApi_APIHANDLE     api_hdl,
   ibApi_GROUPID       network,
   ibApi_UINT32 *      wakeup_interval,
   ibApi_UINT32 *      sleep_interval,
   ibApi_UINT32 *      duty_ratio
);
```

This function retrieves the low power router wakeup interval and duty ratio for all devices.

**Parameters**:

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

*network*: (input) Group ID of the network.

*wakeup_interval*: (output) Wakeup interval in milliseconds.

*sleep_interval:* (output) Sleep time in milliseconds.

*duty_ratio*: (output) the duty ratio (5 - 1000), in increments of 5.

**Return Value**:

ibApi_RESULT_SUCCESS if successful, ibApi_RESULT_ROUNDOFF if round off error detected or other error code (<0) upon failure.

# iBeanAPI_performance.h

## Data Structures

1. **`ibApi_PFM_PacketStats_s`**

```
struct ibApi_PFM_PacketStats_s {
   ibApi_UINT8 ctrlcode;
   ibApi_UINT16 packetTxCount;
   ibApi_UINT16 packetSearchCount;
};
```

**Structure Fields**:

| | |
|---|---|
| *ctrlcode* | Control code. |
| *packetTxCount* | Number of transmissions and acks for this packet. |
| packetSearchCount | Number of search packets and responses caused by this packet. |

2. **`ibApi_PFM_NodeStats_s`**

```
struct ibApi_PFM_NodeStats_s {
   ibApi_UINT8 ctrlcode;
   ibApi_UINT8 reportingFrequency;
   ibApi_UINT16 packetSearchCount;
   ibApi_UINT8 searchPacketTxCount;
   ibApi_UINT16 searchResponseRxCount;
   ibApi_UINT16 searchPacketRxCount;
   ibApi_UINT16 searchResponseTxCount;
   ibApi_UINT16 rejectedPacketCount;
   ibApi_UINT16 acceptedPacketCount;
   ibApi_UINT16 transmittedPacketCount;
   ibApi_UINT16 transmissionsCount;
   ibApi_UINT16 validInterrupts;
   ibApi_UINT32 totalInterrupts;  //2 -> 4
   ibApi_UINT8  numberOfAssociations;
};
```

**Structure Fields**:

| | |
|---|---|
| *ctrlcode* | Control code. |
| *reportingFrequency* | |
| *packetSearchCount* | Number of search packets and responses caused by this packet. |
| *searchPacketTxCount* | Number of search packets transmitted. |
| *searchResponseRxCount* | Number of search packets received. |
| *searchPacketRxCount* | Number of search packets received. |
| *searchResponseTxCount* | Number of search responses transmitted. |
| *rejectedPacketCount* | Number of upstream packets rejected. |
| *acceptedPacketCount* | Number of upstream packets accepted. |
| *transmittedPacketCount* | Number of unique upstream packets. |
| *transmissionsCount* | Number of transmissions (excluding search packets and responses). |
| *validInterrupts* | |
| *totalInterrupts* | 2 - 4 |
| *numberOfAssociations* | |

3. **`ibApi_PFM_Stats_s`**

```
struct ibApi_PFM_Stats_s {
    struct ibApi_PFM_PacketStats_s  packetStats;
    struct ibApi_PFM_NodeStats_s    nodeStats;
};
```

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

**Structure Fields**:

| | |
|---|---|
| *packetStats* | Packet statistics. |
| *nodeStats* | Node statistics. |

## Functions

1. **`ibApi_GetStatisticData`**

```
ibApi_FUNC ibApi_GetStatisticData(
  ibApi_APIHANDLE api_hdl,
  ibApi_DEVICEID device_id,
  ibApi_PFM_STATS *statisticData);
```

This function retrieves the statistical data for the given device.

**Parameters**:

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

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

*device_id:* (output) the statistic data of the device.

**Return Value**:

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

# Example API Code

Millennial Net provides as part of the reference kit, several sample API applications to be compiled under Windows and Linux. This section lists the source code for one of the API examples, ListDevices. ListDevicesprovides a list of all detected network nodes (MeshGate, Mesh Node(s), and End Node(s)).

The C file containing the code shown here can be found in the Programs directory:

**C:\Program Files\MeshScape\examples\windows\ListDevices.c**

```
/*
 * ListDevices.c
 *
 * Copyright (c) 2000-2005 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;
}
```

# A

# Running the Demo Application

This chapter contains information on how to run the demonstration application included with the RK-5424-5 Reference Kit:

## Running the MeshScape Demo Program

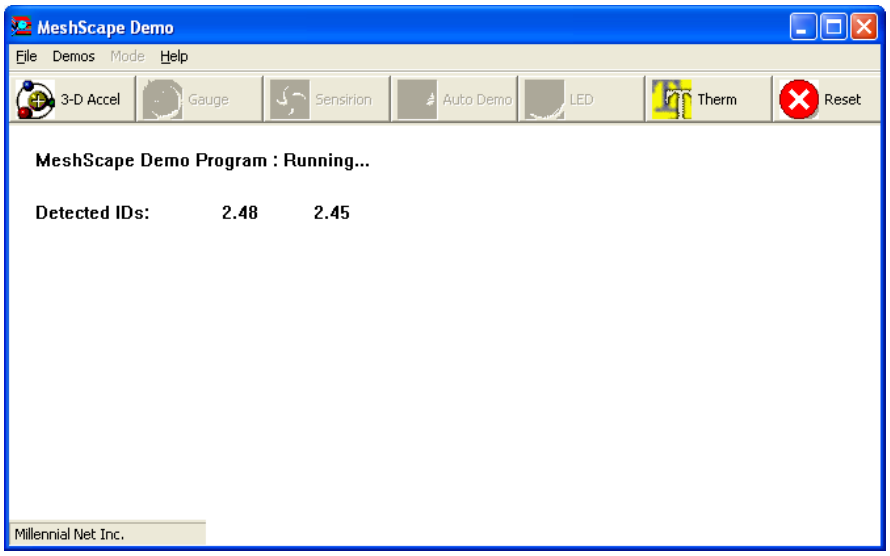
Your RK-5424-5 Reference Kit includes the MeshScape Demonstration program that enables you to run a temperature sensor demo program that demonstrate the features and benefits of the MeshScape system.

To launch the MeshScape Demo program:

- From the Windows taskbar, select:
  **Start>All Programs>MeshScape>API Examples**, then open the **bin** folder and double-click **Demo.exe** to run the executable.

  The MeshScape Demo main window appears as shown in Figure A-1.

**Figure A-1. MeshScape Demo main window**



MeshScape Demo Program : Running...

Detected IDs:      2.48      2.45

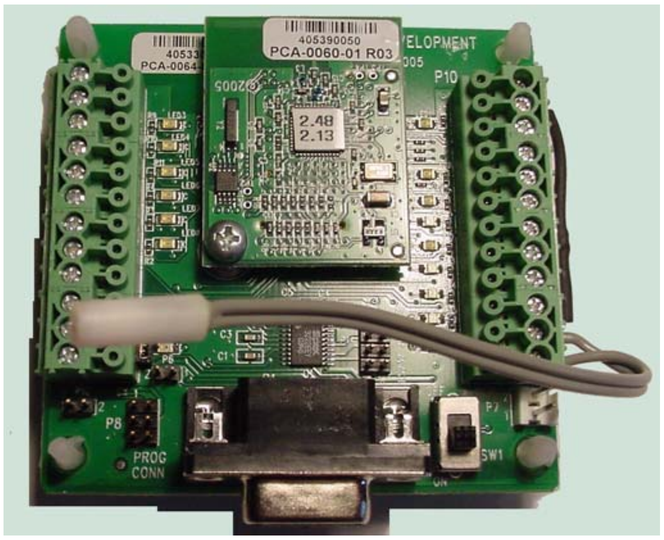
You can close the MeshScape Demo program by selecting **File>Exit** or by clicking the Bye icon on the main window.

Click Reset to close a demonstration application that was launched from within the main window.

# Running the Temperature Sensor Demo

The temperature sensor demo allows you to view a graphical display of a temperature reading from a kele sensor installed on an end node terminal board. Your RK-5424-5 Reference Kit includes an end node terminal board equipped with a kele temperature sensor as shown in Figure A-2.

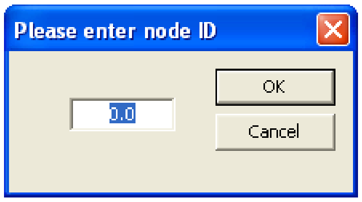
**Figure A-2. End node terminal board with kele temperature sensor**



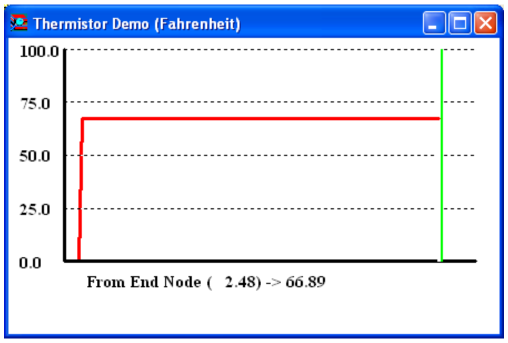
Before launching the temperature sensor demo, turn on the end node device equipped with the sensor, and verify that its Device ID appears in the Detected IDs list in the MeshScape Demo main window.

To launch the Temperature Sensor demo:

1. Select **Demos>Thermistor>Start,** or click the Therm icon.
2. When prompted, enter the Device ID of the node on which the temperature sensor is installed, and then click **OK**.



A window appears displaying a graphic representation of the temperature reading from the node on which you installed the temperature sensor. It may take up to 20 seconds for the node to update its configuration and start to display its temperature reading in the graphical window.



From End Node ( 2.48) -> 66.89

3. To exit the demo, select **Demos>Thermistor>end,** or click the Reset icon.

# B

# Using MeshScape Programmer

The MeshScape Programmer application supplied with your RK-5424-5 Reference Kit enables you to upgrade the firmware and re-configure the group and device IDs on all of your MeshScape devices. This chapter contains information on how to use the MeshScape Programmer application and includes:

# Getting Started with MeshScape Programmer

MeshScape Programmer is a feature-rich application that enables you to upgrade your MeshScape system by:

- re-programming the device flash memory
- unlocking firmware feature upgrades
- re-programming group and device IDs
- re-programming the device radio configuration

To use MeshScape Programmer, you will need to:

- connect the MeshGate to your computer
- connect the target device that you wish to upgrade to your computer
- run the MeshScape Programmer application on your computer

## Connecting the Target Device to Your Computer

Follow the procedures below to connect the target MeshScape device to your computer for programming.

### *Connecting a MeshGate for Programming*

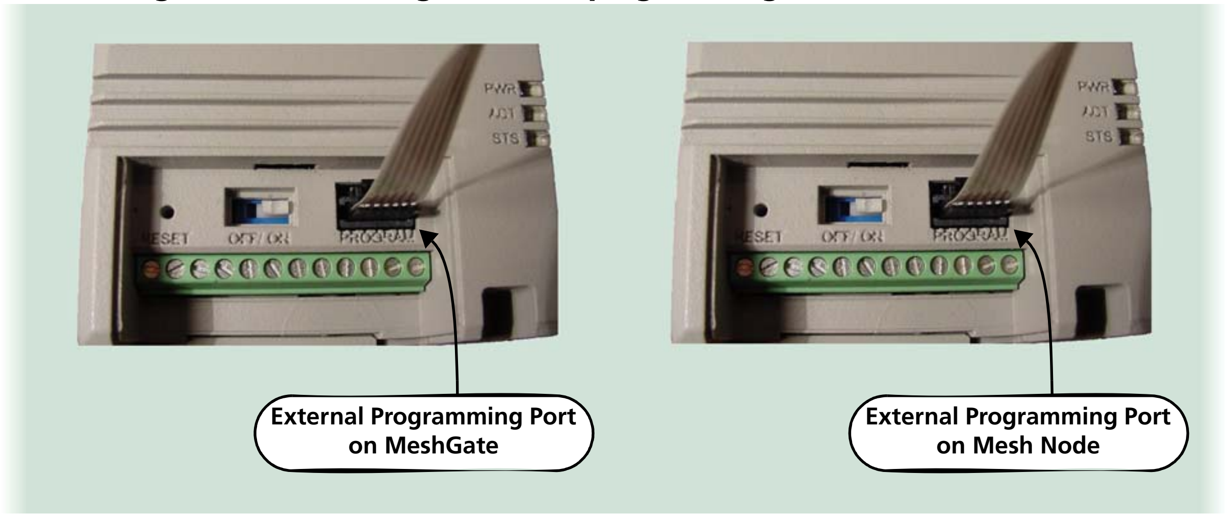
The MeshGate provides a console port equipped with a mini-connector to facilitate firmware upgrades.

Using the mini-connector-to-nine-pin serial connector cable supplied in your RK-5424-5 Reference Kit, connect the MeshGate console port to the serial port on your computer.

### *Connecting a Mesh Node for Programming*

1. Connect a MeshGate to your computer as described above.
2. Remove the connector panel access cover from the MeshGate connected to the computer and remove the connector panel access cover from the target mesh node.
3. Referring to Figure B-1 and using the supplied MeshGate programming cable, connect the **external programming port on the MeshGate** to the **external programming port on the mesh node**.
4. Replace the MeshGate and mesh node connector panel access covers, and disconnect the programming cable after the firmware update is complete.

**Figure B-1.Connecting MeshGate programming cable to a mesh node**

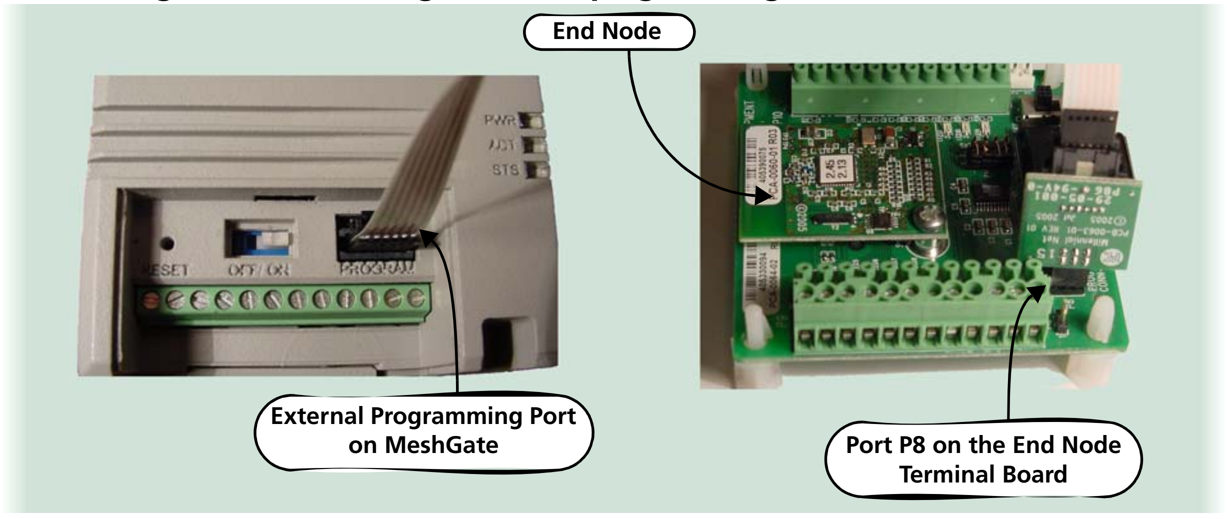

PWR
ACT
STS
RESET
OFF / ON
PROGRAM
External Programming Port on MeshGate
External Programming Port on Mesh Node

### *Connecting an End Node for Programming*

1. Connect a MeshGate to your computer as described on page B-2.
2. Remove the connector panel access cover from the MeshGate connected to the computer.
3. Referring to Figure B-2 and using the supplied MeshGate programming cable and MeshGate-to-end node adapter, connect the **external programming port on the MeshGate** to **port P8 on the end node terminal board**.

   Ensure that Pin 1 on the MeshGate-to-end node adapter mates with Pin1 on connector P8 when making this connection.
4. Replace the MeshGate connector panel access cover, and disconnect the programming cable after the firmware update is complete.

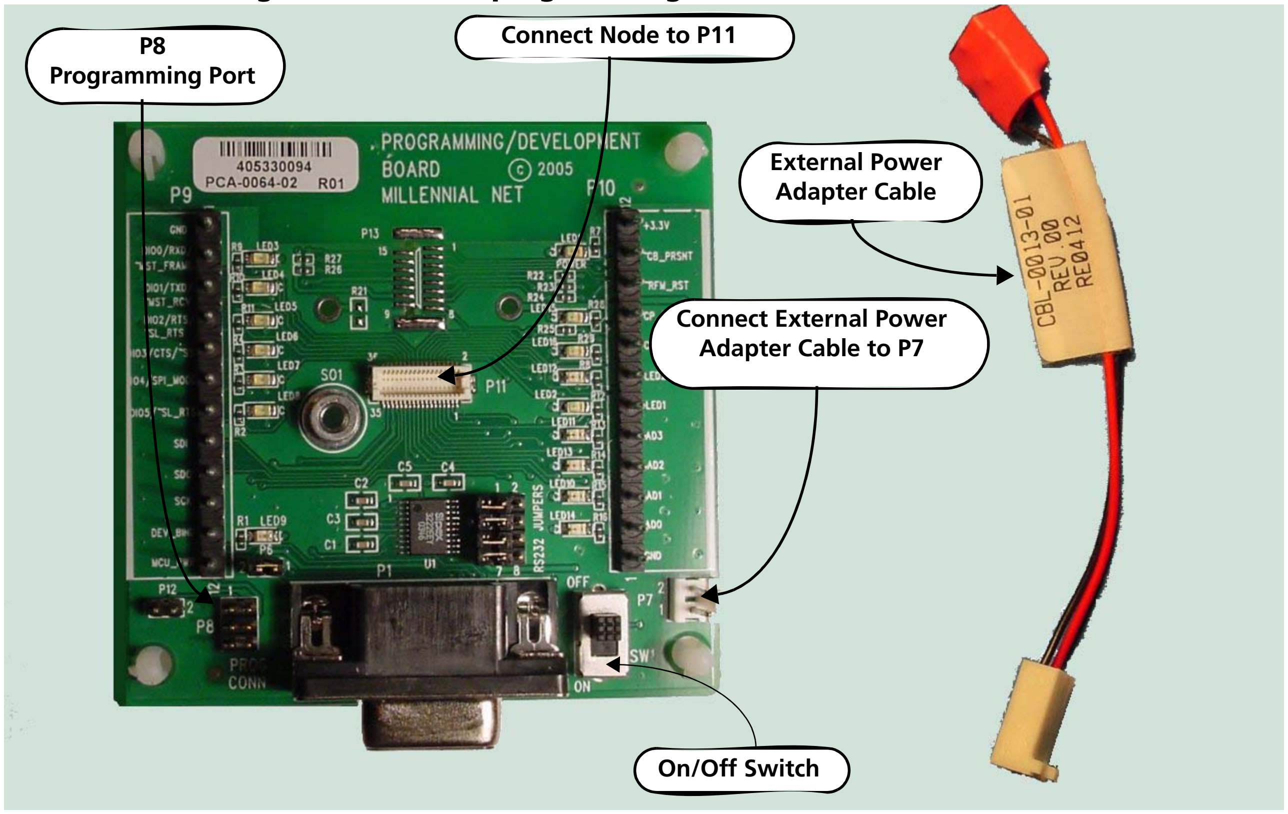
**Figure B-2.Connecting MeshGate programming cable to an end node**


End Node
PWR
ACT
STS
RESET
OFF / ON
PROGRAM
External Programming Port on MeshGate
Port P8 on the End Node Terminal Board

### *Using the Module Programming Terminal Board*

A terminal board is included in your Installation Kit for programming end node or mesh node modules. This module programming terminal board is shown in Figure B-3.

**Figure B-3.Module programming terminal board**



To program an end node or mesh node module using this terminal board:

1. With switch **SW1** in the **Off** position, connect the end node or mesh node module to connector **P11** on the module programming terminal board.
2. Referring to Figure B-2 and using the supplied MeshGate programming cable and MeshGate-to-end node adapter, connect the **external programming port on the MeshGate** to **port P8 on the module programming terminal board**.

   Ensure that Pin 1 on the MeshGate-to-end node adapter mates with Pin1 on connector P8 when making this connection.
3. Remove the 3.3v lithium battery on the back side of the module programming terminal board.
4. Connect the external power adapter cable supplied in your Installation Kit to connector **P7** on the module programming terminal board.
5. Connect the supplied AC adapter to the external power adapter cable, and then plug the AC adapter into a 110/220 VAC power source.
6. Move switch **SW1** to the **On** position.

7. Run the MeshScape Programmer application as described in the next section.
8. Move switch **SW1** on the terminal board to the **Off** position.
9. Remove the programmed node from the terminal board.
10. Repeat steps 1 to 8 until all nodes have been configured as desired.

## Launching MeshScape Programmer Using Windows

To launch MeshScape Programmer, do one of the following:

- Double-click on the desktop's MeshScape Programmer icon.
- From the Windows taskbar, select:
  **Start>All Programs>MeshScape>MeshScape Programmer**.

MeshScape Programmer runs as described in the next section.

# Performing MeshScape Programmer Operations

You may use MeshScape Programmer to:

- upgrade the firmware image on the target device
- unlock features on the target device
- re-program the device and group IDs on the target device
- re-program the radio configuration on the target device

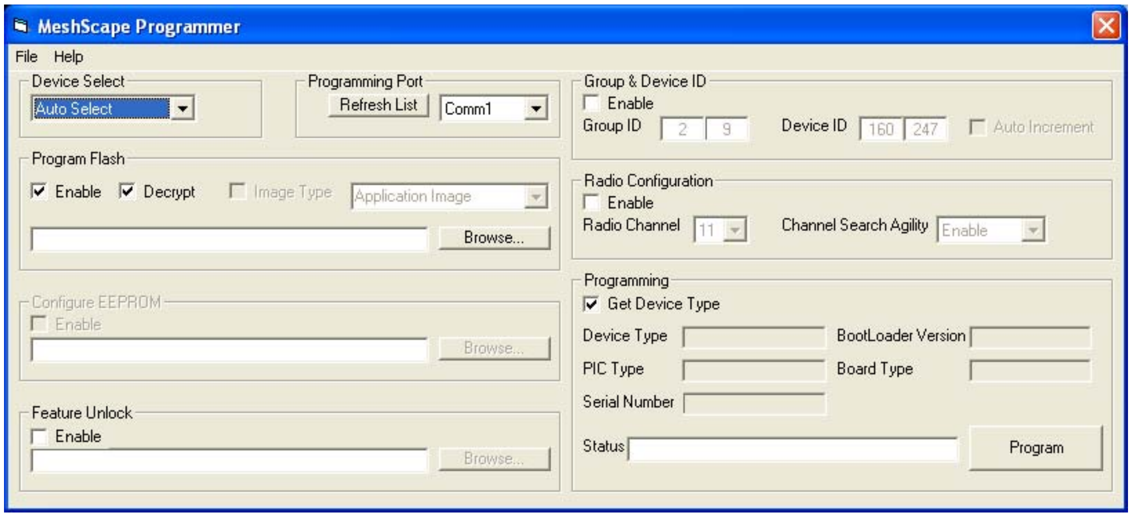
## Upgrading Firmware on the Target Device

To upgrade the firmware on a MeshGate, mesh node, or end node using the MeshScape Programmer application:

1. Connect the target device to your computer and launch the MeshScape Programmer application as described in the previous section.

   The MeshScape Programmer main window appears as shown in Figure B-4.

**Figure B-4.The MeshScape Programmer main window**



MeshScape Programmer

File Help

Device Select: Auto Select

Programming Port: Refresh List | Comm1

Program Flash: ✓ Enable ✓ Decrypt ☐ Image Type | Application Image | Browse...

Configure EEPROM: ☐ Enable | Browse...

Feature Unlock: ☐ Enable | Browse...

Group & Device ID: ☐ Enable | Group ID 2 9 | Device ID 160 247 | ☐ Auto Increment

Radio Configuration: ☐ Enable | Radio Channel 11 | Channel Search Agility Enable

Programming: ✓ Get Device Type | Device Type | BootLoader Version | PIC Type | Board Type | Serial Number | Status | Program

2. Select the target device to program by selecting an option from the **Device Select** drop-down menu:
   - Auto-Select - Allow MeshScape Programmer to determine the connected device type (default).
   - MeshGate Application - Upgrade the firmware on the MeshGate's main application terminal board.
   - MeshGate RF - Upgrade the firmware on the MeshGate's RF daughter board.
   - External Device - Upgrade a mesh node or end node connected to the MeshGate.

3. Select the PC serial port to use when communicating with the connected MeshGate from the **Programming Port** drop-down menu.

   If a required Comm port is being used by another application, close the application and then click **Refresh List** to make the Comm port available for use by MeshScape Programmer.

4. Select the firmware image file to load on the target device.

   a. Mark the **Enable** checkbox to enable the program flash operation.

   b. Mark the **Decrypt** checkbox to decrypt an encrypted image file. Encrypted image files are denoted by a .enc file extension.

   c. Click the **Browse** button, and select the firmware image file supplied to you from Millennial Net.

5. Upload the new image file to the target device.

   a. Mark the **Get Device Type** checkbox to display information about the target device in the Device Type field as the upgrade progresses.

   b. Click **Program** to initiate the firmware upgrade.

As the firmware upgrade progresses, you will see status messages posted to the **Status** field. Once the upgrade is completed, the status will be reported as Programming Complete.

## Unlocking Features on the Target Device

To unlock features on a MeshGate, mesh node, or end node using the MeshScape Programmer application:

1. Connect the target device to computer and launch the MeshScape Programmer application as described on page B-2.

   The MeshScape Programmer main window appears as shown in Figure B-4.

2. Select the target device to program by selecting an option from the **Device Select** drop-down menu:
   – Auto-Select - Allow MeshScape Programmer to determine the connected device type (default).
   – MeshGate Application - Upgrade the firmware on the MeshGate's main application terminal board.
   – MeshGate RF - Upgrade the firmware on the MeshGate's RF daughter board.
   – External Device - Upgrade a mesh node or end node connected to the MeshGate.

3. Select the PC serial port to use when communicating with the connected MeshGate from the **Programming Port** drop-down menu.

   If a required Comm port is being used by another application, close the application and then click **Refresh List** to make the Comm port available for use by MeshScape Programmer.

4. De-select (i.e., uncheck) the Enable and Decrypt program flash options.

5. Select the feature unlock file to load on the target device.
    a. Mark the **Enable** checkbox to enable the feature unlock operation.
    b. Click the **Browse** button, and select the feature unlock file supplied to you from Millennial Net.
6. Upload the feature unlock file to the target device.
    a. Mark the **Get Device Type** checkbox to display information about the target device in the Device Type field as the upgrade progresses.
    b. Click **Program** to initiate upload of the feature unlock file to the target device.

As the feature unlock file upload progresses, you will see status messages posted to the **Status** field. Once the update is completed, the status will be reported as Programming Complete.

## Reprogramming the Group and Device IDs on the Target Device

To reprogram the group and device IDs on a MeshGate, mesh node, or end node using the MeshScape Programmer application:

1. Connect the target device to computer and launch the MeshScape Programmer application as described on page B-2.

   The MeshScape Programmer main window appears as shown in Figure B-4.
2. Select the target device to program by selecting an option from the **Device Select** drop-down menu:
   – Auto-Select - Allow MeshScape Programmer to determine the connected device type (default).
   – MeshGate Application - Upgrade the firmware on the MeshGate's main application terminal board.
   – MeshGate RF - Upgrade the firmware on the MeshGate's RF daughter board.
   – External Device - Upgrade a mesh node or end node connected to the MeshGate.
3. Select the PC serial port to use when communicating with the connected MeshGate from the **Programming Port** drop-down menu.

   If a required Comm port is being used by another application, close the application and then click **Refresh List** to make the Comm port available for use by MeshScape Programmer.
4. De-select (i.e., uncheck) the Enable and Decrypt program flash options.
5. Enter the group and device IDs to set on the target device.
    a. Mark the **Enable** checkbox to enable the ID reprogramming.
    b. Enter the new IDs in the **Group ID** and **Device ID** fields.
    c. Optional. Mark the **Auto Increment** checkbox if you will be reprogramming the group and device IDs on multiple devices and wish to do so in sequence.
6. Upload the new group and device IDs to the target device.

a. Mark the **Get Device Type** checkbox to display information about the target device in the Device Type field as the upgrade progresses.

b. Click **Program** to initiate upload of the new IDS to the target device.

As the group and device ID reprogramming progresses, you will see status messages posted to the **Status** field. Once the update is completed, the status will be reported as Programming Complete.

## Reprogramming the Target Device's Radio Configuration

To reprogram the radio configuration on a MeshGate, mesh node, or end node using the MeshScape Programmer application:

1. Connect the target device to computer and launch the MeshScape Programmer application as described on page B-2.

   The MeshScape Programmer main window appears as shown in Figure B-4.

2. Select the target device to program by selecting an option from the **Device Select** drop-down menu:
   – Auto-Select - Allow MeshScape Programmer to determine the connected device type (default).
   – MeshGate Application - Upgrade the firmware on the MeshGate's main application terminal board.
   – MeshGate RF - Upgrade the firmware on the MeshGate's RF daughter board.
   – External Device - Upgrade a mesh node or end node connected to the MeshGate.

3. Select the PC serial port to use when communicating with the connected MeshGate from the **Programming Port** drop-down menu.

   If a required Comm port is being used by another application, close the application and then click **Refresh List** to make the Comm port available for use by MeshScape Programmer.

4. De-select (i.e., uncheck) the Enable and Decrypt program flash options.

5. Define the radio configuration on the target device.

   a. Mark the **Enable** checkbox to enable radio configuration reprogramming.

   b. Select the radio channel on which the device is to communicate within the MeshScape from the **Radio Channel** drop-down menu. Valid channel selections are 11 to 26. Ensure that all devices in the MeshScape are communicating on the same channel.

      Optional. Select **Enable** from the **Channel Search Agility** drop-down menu to enable the channel search agility feature.

      Channel search agility is a means by which a MeshScape device can detect when the MeshGate's communication channel is changed or lost, and then search for a new channel on which to re-establish communication with the MeshGate.

6. Upload the new radio configuration to the target device.

   a. Mark the **Get Device Type** checkbox to display information about the target device in the Device Type field as the upgrade progresses.

   b. Click **Program** to initiate upload of the new radio configuration to the target device.

As the radio configuration reprogramming progresses, you will see status messages posted to the **Status** field. Once the update is completed, the status will be reported as Programming Complete.

# Glossary

| | |
|---|---|
| **API** | Application Programming Interface: A set of definitions of the ways in which one piece of computer software communicates with another. It is a method of achieving abstraction, usually (but not necessarily) between lower-level and higher-level software. One of the primary purposes of an API is to provide a set of commonly used functions-for example, to poll a wireless network for active network nodes (mesh nodes and end nodes). Programmers can then take advantage of the API by making use of its functionality, saving them the task of programming everything from scratch. APIs themselves are abstract: software that provides a certain API is often called the implementation of that API. |
| **ad hoc network** | A group of wireless sensors connected as an independent wireless network, communicating directly with each other without the use of a mesh node. |
| **bandwidth** | The amount of data that can be transmitted in a fixed amount of time. For digital devices, the bandwidth is usually expressed in bits per second (bps) or bytes per second. For analog devices, the bandwidth is expressed in cycles per second, or Hertz (Hz). |
| **data model** | As it pertains to wireless sensor networks, the data model characterizes and describes the way in which data flows through and is used in the network. Common data model categories include data collection models (periodic sampling, event driven, and store and forward) and bi-directional dialogue data models (polling and on demand). |
| **DSSS** | Direct Sequence Spread Spectrum: Spread spectrum method of spreading a narrow band signal. This method uses special pseudo noise codes to expand the narrow band signal out across a broad portion of the radio band. (See also *FHSS* and *spread spectrum*.) |
| **duty cycle** | The duty cycle of a module refers to the percentage of time the module is active versus inactive. |
| **end node** | The network module that provides the physical interface between the wireless sensor network and the sensor or actuator that it is wired to. Sometimes called a Reduced Function Device (see *RFD*). |
| **endpoint** | See *end node*. |
| **FFD** | Full Function Device: A term referring to a device that can act as an intermediate mesh node, passing data from other devices. (See also *RFD*.) |

| | |
|---|---|
| **FHSS** | Frequency Hopping Sequence Spread Spectrum: Spread spectrum method of spreading a narrow band signal out across a broad portion of the radio band. This method "hops" the signal as a function of time. (See also *DSSS* and *spread spectrum*.) |
| **gateway** | The network module that provides the interface between the application platform and the modules on the wireless sensor network. |
| **IEEE** | Institute of Electrical and Electronics Engineers: Organization of engineers, scientists, and students that is known for developing standards for the computer and electronics industry. |
| **IEEE 802.15.4** | Standard developed by IEEE that defines the lower protocol layers (PHY and MAC) for low-data-rate wireless Personal Area Networks (PANs). |
| **ISM** | The industrial, scientific, and medical (ISM) radio bands were originally reserved internationally for non-commercial use of RF electromagnetic fields for industrial, scientific and medical purposes. They are now also used for license-free error-tolerant wireless communications applications in the 900 MHz and 2.4 GHz bands. |
| **latency** | In networking, the amount of time it takes a packet to travel from source to destination. Together, latency and bandwidth define the speed and capacity of a network. |
| **mesh node** | The module on the wireless sensor network used to extend network coverage area, route around obstacles, and provide back-up routes in case of network congestion or device failure. The mesh node can also provide a direct physical interface to a sensor or actuator. Sometimes called a Full Function Device (see *FFD*). |
| **mesh topology** | A wireless sensor networking architecture consisting of a gateway and mesh nodes that provides extended area coverage, routing around obstacles, and back-up data paths. |
| **narrowband** | Radio signal that contains all of its power within a very narrow portion of the radio frequency band. |
| **OSI** | Open System Interconnection: An ISO standard for worldwide communications that defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, proceeding to the bottom layer, over the channel to the next station and back up the hierarchy. |
| **packet** | A piece of a message transmitted over a packet-switching network. One of the key features of a packet is that it contains the destination address in addition to the data. |

| | |
|---|---|
| **personal area network** | A personal area network (PAN) is the interconnection of information technology devices within the range of an individual person, typically within a range of 10 meters. |
| **protocol** | The protocol defines a common set of rules and signals that devices (nodes) on the network use to communicate. |
| **protocol stack** | A set of network protocol layers that work together. The OSI reference model that defines seven protocol layers is often called a stack. |
| **RFD** | Reduced Function Device: A term referring to a device that is just smart enough to talk to the network (see also *FFD*). |
| **router** | See *mesh node*. |
| **sensor node** | A wireless sensor network node consisting of a sensor or actuator device attached to a wireless module. The wireless module provides the interface between the sensor device and the wireless network. |
| **SNR** | Signal-to-noise ratio: the ratio of the amplitude of a desired analog or digital data signal to the amplitude of noise in a transmission channel at a specific point in time. SNR is typically expressed logarithmically in decibels (dB). SNR measures the quality of a transmission channel or an audio signal over a network channel. The greater the ratio, the easier it is to identify and subsequently isolate and eliminate the source of noise. A SNR of zero indicates that the desired signal is virtually indistinguishable from the unwanted noise. |
| **spread spectrum (wideband)** | Technique for taking a narrowband signal and spreading it across a broader portion of the radio frequency band. Spread-spectrum signals are more resistant to interference than narrow band signals. The two basic methods for spreading a narrowband are direct sequence and frequency hopping. (See also *DSSS* and *FHSS*.) |
| **star topology** | A wireless sensor networking architecture consisting of a gateway and end nodes that is extremely power efficient for short-range networks. |
| **star-mesh hybrid topology** | A wireless sensor networking architecture consisting of a gateway, mesh nodes, and end nodes that optimizes range and power efficiency of the network. |
| **topology** | As it pertains to wireless sensor networks, the geometric arrangement of the modules (gateway, mesh nodes, and end nodes) within a network. Common topologies include star, mesh, and star-mesh hybrid. |

# Index