SmartMesh WirelessHART User's Guide

Table of Contents

1 About This Guide

1.1 Related Documents

The following documents are available for the SmartMesh WirelessHART network:

Getting Started with a Starter Kit

- SmartMesh WirelessHART Easy Start Guide - walks you through basic installation and a few tests to make sure your network is working
- SmartMesh WirelessHART Tools Guide - the Installation section contains instructions for the installing the serial drivers and example programs used in the Easy Start Guide and other tutorials.

User Guide

- SmartMesh WirelessHART User's Guide - describes network concepts, and discusses how to drive mote and manager APIs to perform specific tasks, e.g. to send data or collect statistics. This document provides context for the API guides.

Interfaces for Interaction with a Device

- SmartMesh WirelessHART Manager CLI Guide - used for human interaction with a Manager (e.g. during development of a client, or for troubleshooting). This document covers connecting to the CLI and its command set.
- SmartMesh WirelessHART Manager API Guide - used for programmatic interaction with a manager. This document covers connecting to the API and its command set.
- SmartMesh WirelessHART Mote CLI Guide - used for human interaction with a mote (e.g. during development of a sensor applicaition, or for troubleshooting). This document covers connecting to the CLI and its command set.
- SmartMesh WirelessHART Mote API Guide - used for programmatic interaction with a mote. This document covers connecting to the API and its command set.

Software Development Tools

- SmartMesh WirelessHART Tools Guide - describes the various evaluation and development support tools included in the SmartMesh SDK including tools for exercising mote and manager APIs and visualizing the network.

Application Notes

- SmartMesh WirelessHART Application Notes - app notes covering a wide range of topics specific to SmartMesh WirelessHART networks and topics that apply to SmartMesh networks in general.

Documents Useful When Starting a New Design

- The Datasheet for the LTC5800-WHM SoC, or one of the castellated modules based on it, or the backwards compatible LTP5900 22-pin module.
- The Datasheet for the LTP5903-WHR embedded manager.
- A Hardware Integration Guide for the mote SoC or castellated module, or the 22-pin module - this discusses best practices for integrating the SoC or module into your design.
- A Hardware Integration Guide for the embedded manager - this discusses best practices for integrating the embedded manager into your design.
- A Board Specific Integration Guide - For SoC motes and Managers. Discusses how to set default IO configuration and crystal calibration information via a "fuse table".
- Hardware Integration Application Notes - contains an SoC design checklist, antenna selection guide, etc.
- The ESP Programmer Guide - a guide to the DC9010 Programmer Board and ESP software used to program firmware on a device.
- ESP software - used to program firmware images onto a mote or module.
- Fuse Table software - used to construct the fuse table as discussed in the Board Specific Integration Guide.

Other Useful Documents

- A glossary of wireless networking terms used in SmartMesh documentation can be found in the SmartMesh WirelessHART User's Guide.
- A list of Frequently Asked Questions

1.2 Conventions Used

The following conventions are used in this document:

`Computer type` indicates information that you enter, such as specifying a URL.

Bold type indicates buttons, fields, menu commands, and device states and modes.

Italic type is used to introduce a new term, and to refer to APIs and their parameters.

✓ Tips provide useful information about the product.

ℹ Informational text provides additional information for background and context

⚠ Notes provide more detailed information about concepts.

⛔ Warning! Warnings advise you about actions that may cause loss of data, physical harm to the hardware or your person.

```
code blocks display examples of code
```

1.3 Revision History

Revision	Date	Description
1	07/17/2012	Initial release
2	03/18/2013	Numerous small changes
3	10/22/2013	Added description of Linux HA hooks, other minor corrections
4	04/04/2014	Added details on the temperature data generator for master mode;
5	10/28/2014	Detailed compliance for EN 300 328 rev. 1.8.2; Updated description of OTAP; Other minor changes
6	04/22/2015	Updated blacklisting requirements; Added disconnected state; Other minor changes
7	12/03/2015	Minor changes

2 SmartMesh Glossary

Access point (or AP) - The device that bridges the wireless and wired networks. Converts wireless MAC packets to wired Net packets and vice versa.

Acknowledgement (or ACK) - A frame sent in response to receiving a packet, confirming that the packet was properly received. The ACK contains the time difference between the packet receiver and sender.

Advertisement - A frame sent to allow other devices to synchronize to the network and containing link information required for a new mote to join.

ASN - Absolute Slot Number. The number of timeslots that have elapsed since the start of the network (WirelessHART) or 20:00:00 UTC July 2,2002 (IP if UTC is set before starting the network)

Authentication - The cryptographic process of ensuring that the packet received has not been modified, and that it originated from the claimed net layer sender.

Availability - The fraction of user packets accepted (ACK'd) by the mote API. An persistent availability < 1 means that user packet latency is increased due to full mote queues.

Backbone - A feature in a SmartMesh network that allows motes to share a fast superframe to enable low-latency alarm traffic.

Bandwidth - The capacity of a mote to transmit data, usually expressed in packets/s.

Base Bandwidth - The bandwidth each mote in a network gets without having to request a service.

CCM* - **C**ounter mode with **C**BC-**M**AC. The authentication/encryption scheme used in Dust products. See the application note "SmartMesh Security" for a more detailed description,

Channel - The index into a list of center frequencies used by the PHY. In 802.15.4, there are 16 channels in the 2.4-2.4835 GHz Instrumentation, Scientific, and Medical (ISM) band.

Channel Hopping - Changing channel between slots, also called "slow hopping" to distinguish from PHY's that change channel within a message, such as Bluetooth.

Channel Offset - A link-specific number used in the channel calculation function to pick which channel to use in this ASN.

Child - A device that receives time information from another mote is its child. A child forwards data through its parent.

CSMA - Carrier Sense Multiple Access. A communications architecture where an unsynchronized transmitter first senses if others are transmitting before attempting its transmission.

Commissioning - The act of configuring motes for use in a deployment, typically by setting Network ID, join key, and other joining parameters.

Discovery - The process by which motes find potential neighbors, and report that information back to the manager.

Downstream - The direction away from the manager or wired-side application and into the mesh network.

Encryption - The cryptographic process of converting payload information into a form indistinguishable from random noise, such that it can only be read by the intended recipient.

Frame (or packet) - Information sent by the PHY layer. Typically containing per-hop (MAC) and end-to-end (Net) layer headers and a payload.

Gateway - The device in a WirelessHART network responsible for abstracting the wired HART data model from the WirelessHART mesh implementation.

Graph - A numbered routing element included in the WirelessHART net and IP mesh layer headers which tells a mote where to send each packet. Superframes and slotframes also have a graph ID.

Graph route - A route where only the graph ID is specified in the packet header. A packet can follow the graph route through multiple motes to its destination. Compare with source route.

Guard time - The time a device listens in advance or after the expected packet arrival time to allow for imperfect synchronization between devices.

Health report - A packet sent by a mote conveying its internal state and the quality of its neighbor paths. Health reports are used by the manager in optimization and diagnostics.

Idle listen - A slot where the receiving device wakes up to receive a packet but the transmitting device does not send one. About two-thirds of all receive slots end up as idle listens in a typical network.

Joining - The sequence of handshakes between a new mote and a manager to bring the mote into the network. It begins with a mote presenting an encrypted request and ends with link and run-time security credential assignment.

Keep alive - An empty packet sent to keep devices synchronized after a timeout during which no data packets have been sent on a path. This timeout is typically 30 seconds in WirelessHART and 15 seconds in IP.

Latency - The time difference between packet generation and arrival at its final destination.

Leaf - A mote that presently has no upstream RX links, and thus no children. Also called a leaf node or leaf mote.

Low Power Border Router (LBR) - A logical or physical device which converts 6LowPAN (IPv6 for Low power Wireless Personal Area Networks, RFC 4944) packets into IPv6 packets. Also called an Edge Router.

MAC - The Medium Access Control layer. Technically a sublayer of the Data Link Layer (OSI layer 2), but typically used interchangeably in Dust documentation.

Manager - The device or process responsible for establishing and maintaining the network.

Master (mode) - a mode of mote operation where the mote automatically joins the network for which it was configured, the serial API is disabled, and a resident application allows for interacting with some onboard I/O.

Mesh - A network topology where each mote may be connected to one or more motes.

Mote - A device which provides wireless communications for a field device to transmit sensor or other data. The basic building block of a network.

Multi-hop - A network where one or more motes has no path to the access point. Data packets may sometimes take multiple hops from source to destination.

Multipath - A radio signal that is the superposition of many reflected signals. Used as an adjective to describe phenomenon where signal level can vary dramatically with small environmental changes, *i.e.* multipath propagation, or multipath fading.

NACK - A special ACK frame sent in response to receiving a packet, stating that it was correctly received but NOT accepted for forwarding.

Neighbor - A mote in range of the mote in question.

Non-routing - A mote that has been configured to not advertise. A non-routing mote never forwards packets along a graph or has children.

Optimization - The manager process of taking health report information and using it to modify the network to minimize energy consumption and latency.

Packet - Also called a frame. The variable sized unit of data exchange.

Parent (or time parent) - A device that serves as a source of time synchronization. In Dust networks, a mote's parent is one hop closer to the AP. A parent forwards a child's data towards the manager.

Path - The potential connection between two motes. A path that has assigned links is a used path. One that has been discovered but has no links is an unused path.

Path stability - The ratio of acknowledged packets to sent packets between two motes. Each of the two motes keeps a separate count of the path stability denoted by A->B and B->A. A path where the two motes have significantly different counts of path stability is called an asymmetric path.

Pipe - A feature in a SmartMesh WirelessHART network that enables rapid publishing to and/or from a single target mote.

Provisioning - The number of links assigned by the manager per packet generated by the mote to allow for imperfect stability. Default provisioning is 3x meaning that on average each packet has three chances to be successfully transmitted before the mote starts to accumulate packets.

Publishing rate (Burst rate) - The rate at which an mote application transmits upstream data. In WirelessHART the term burst rate is equivalent to Publishing rate.

PHY - The physical layer, *i.e.* the radio.

Reliability - The percentage of unique packets received relative to the number generated.

Route - The motes that a packet passes through between source and destination, *e.g.* a packet from mote 3 might use the route 3-2-6-AP. Because of the graph routing used upstream, packets originating at the same mote randomly take a variety of routes.

Schedule - The collection of superframes and links in the network, particularly those organized for a particular purpose.

Services (or Timetables) - The process of requesting and receiving (or not) task-specific bandwidth.

Slave (mode) - a mode of mote operation where the mote requires an application to drive its API in order to join the network.

Slotframe or Superframe - A collection of timeslots with a particular repetition period (length) and labeled with a Graph ID.

Source route - A route where every hop between source and destination is explicitly specified in the packet header. Dust networks only use source routing for downstream packets.

Star - A network topology where all motes only have a connection to the the access point (in ZigBee, the PAN coordinator) and are non-routing leafs.

Star-mesh - A network topology where motes form stars around routers, where the routers may have one or more neighbors through whom they can forward mote data.

Statistics - The aggregated information about network topology and performance constructed from the raw mote health reports.

TDMA - Time Division Multiple Access. A communications architecture where packetized information exchange only occurs within timeslots and channel offsets that are assigned exclusively to a pair of devices.

Timeslot - A defined period of time just long enough for a pair of motes to exchange a maximum-length packet and an acknowledgement. Time in the network is broken up into synchronized timeslots.

Upstream - The direction towards the manager or wired-side application from the mesh network.

ZigBee - A single channel, CSMA network protocol.

3 The SmartMesh WirelessHART Network

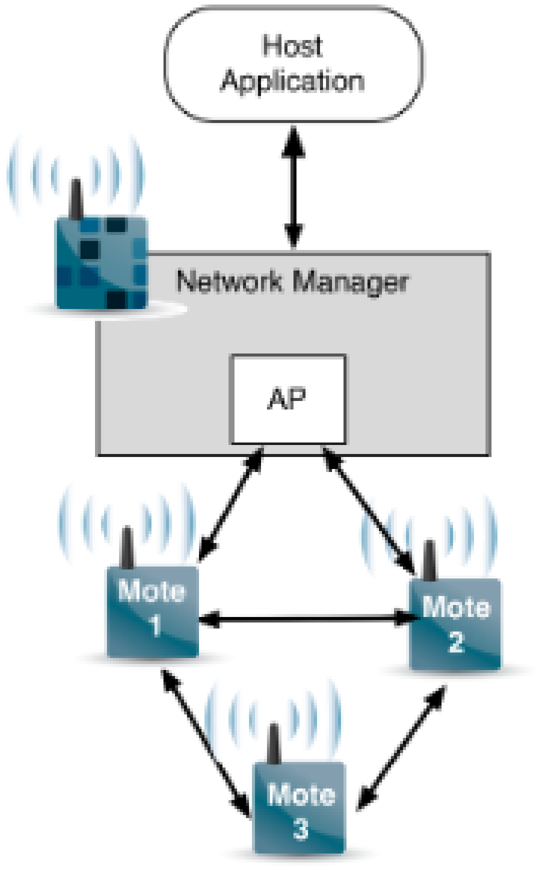
3.1 Introduction

The Network User's Guide is intended to explain fundamental network and device behavior and features at a high level. For details on the APIs referenced, see the SmartMesh WirelessHART Manager Guides and Mote Guides.

3.1.1 Network Overview

A SmartMesh® network consists of a self-forming multi-hop, mesh of nodes, known as *motes*, which collect and relay data, and a *Network Manager* that monitors and manages network performance and security, and exchanges data with a host application.

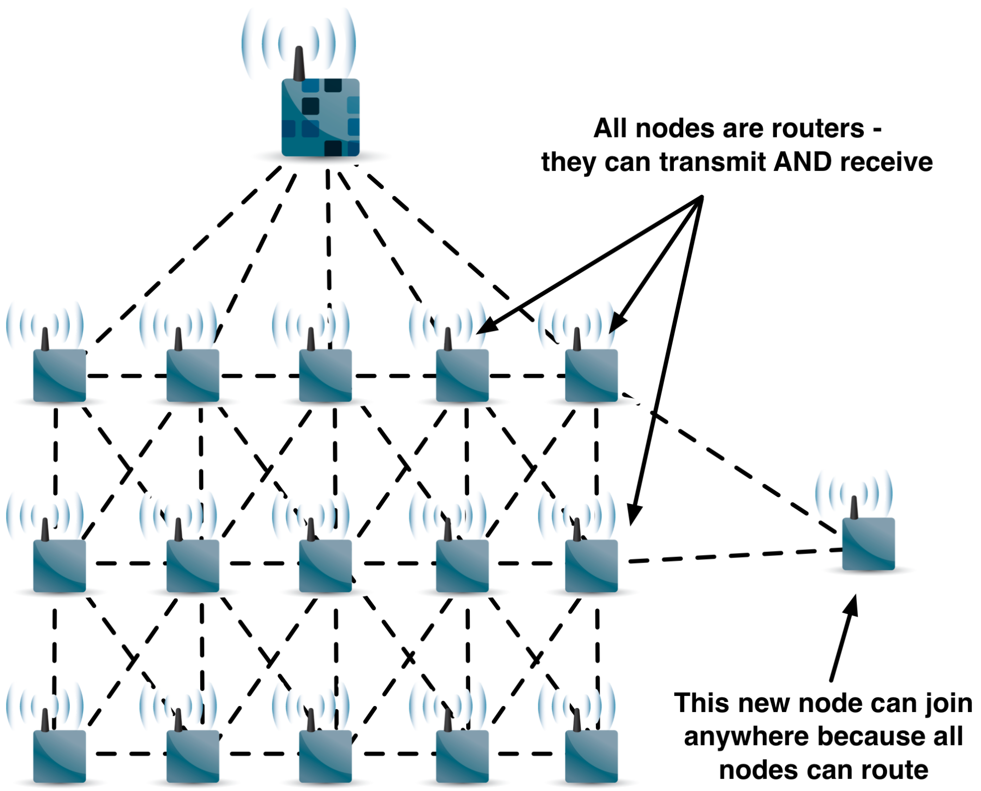
SmartMesh networks communicate using a Time Slotted Channel Hopping (TSCH) link layer, pioneered by Linear's Dust Networks group. In a TSCH network, all motes in the network are synchronized to within less than a millisecond. Time in the network is organized into timeslots, which enables collision-free packet exchange and per-transmission channel-hopping. In a SmartMesh network, every device has one or more *parents* (*e.g.* mote 3 has motes 1 and 2 as parents) that provide redundant *paths* to overcome communications interruption due to interference, physical obstruction or multi-path fading. If a packet transmission fails on one path, the next retransmission may try on a different path and different RF channel. Building networks with sufficient redundancy requires following some simple deployment guidelines - these are outlined in the application note "Planning a Deployment."



A network begins to form when the network manager instructs its on-board access point radio (AP) to begin sending *advertisements* - packets that contain information that enables a device to synchronize to the network and request to join. This message exchange is part of the security handshake that establishes encrypted communications between the manager or application, and mote. Once motes have joined the network, they maintain synchronization through time corrections when a packet is acknowledged.

An ongoing *discovery* process ensures that the network continually discovers new paths as the RF conditions change. In addition, each mote in the network tracks performance statistics (*e.g.* quality of used paths, and lists of potential paths) and periodically sends that information to the network manager in packets called *health reports*. The Network Manager uses health reports to continually optimize the network to maintain >99.999% data reliability even in the most challenging RF environments.

The use of TSCH allows SmartMesh devices to sleep in-between scheduled communications and draw very little power in this state. Motes are only active in timeslots where they are scheduled to transmit or receive, typically resulting in a duty cycle of <1%. The optimization software in the Network Manager coordinates this schedule automatically. When combined with the Eterna low-power radio, every mote in a SmartMesh network – even busy routing ones – can run on batteries for years. By default, all motes in a network are capable of routing traffic from other motes, which simplifies installation by avoiding the complexity of having distinct routers vs. non-routing end nodes. Motes may be configured as *non-routing* to further reduce that particular mote's power consumption and to support a wide variety of network topologies.


All nodes are routers - they can transmit AND receive
This new node can join anywhere because all nodes can route

At the heart of SmartMesh motes and network managers is the Eterna IEEE 802.15.4e System-on-Chip (SoC), featuring our highly-integrated, low power radio design, plus an ARM® Cortex™-M3 32-bit microprocessor running SmartMesh networking software. The SmartMesh software comes fully compiled yet is configurable via a rich set of application programming interfaces (APIs) which allows a host application to interact with the network, e.g. to transfer information to a device, to configure data publishing rates on one or more motes, or to monitor network state or performance metrics. Data publishing can be uniform or different for each device, with motes being able to publish infrequently or faster than once per second as needed.

3.1.2 SmartMesh Network Features

SmartMesh networks provide a simple, reliable way to monitor and control processes and equipment. Using redundant, multi-hop networking and ultra low-power hardware, SmartMesh networks offer unprecedented access to information about the physical world.

SmartMesh networks are:

- *Easy to Install* —They are self-configuring, battery-powered networks that require no site survey or wireless expertise to install.

Benefit: You can deploy a SmartMesh Network within hours, not days.

- *Reliable*— They provide redundant, self-healing routing that approaches the reliability of a wired network.

Benefit: You have the reliability of a wired network with the flexibility of wireless.

- *Manageable* —They provide network-wide quality-of-service metrics and control commands that simplify network management.

Benefit: You can manage multiple networks from a single PC. No device-level coding or management is needed.

The SmartMesh WirelessHART Manager combines Dust Networks' robust intelligent networking and industry-leading low-power radio technology to achieve the high data reliability, lower latency, and deterministic power management required for condition monitoring applications. The SmartMesh WirelessHART Manager acts as both a HART gateway and network manager for SmartMesh WirelessHART motes, creating a self-configuring, reliable wireless mesh network.

The SmartMesh WirelessHART Starter Kit (DC9007) comes standard with one SmartMesh manager and 5 motes. You can configure, monitor, and manage your networks from a PC using onboard web-based administrative tools. Motes in the kits ship in a standalone **master** mode - they contain a sample application that controls joining and can interact with onboard sensors. In the production default **slave** mode, the mote expects an application to drive its API. This is discussed in more detail in the mote section of this guide.

3.2 Network Formation

For a mote to join a network, it needs to get time synchronized to the rest of the network. This is achieved by hearing an *advertisement* from a mote or Access Point (AP) already in the network. The network starts forming when the manager instructs the access point mote to begin sending advertisements. One mote will hear the AP advertisement, then join, and start advertising itself. This process repeats in parallel as other motes join and begin their own advertising. The advertisements are WirelessHART advertisement frames that contain synchronization and link information. In addition to synchronizing the new device, the advertisement also describes when the new device can send in a request to join the network, and when it should expect a reply. This results in temporary links being assigned to the joining mote that it will use until it gets its specific links from the manager.

A WirelessHART manager advertises every 160 ms. Motes within radio range of the AP will join after they have heard one of these advertisements. Average synchronization time for the first-hop motes, at a 5% duty cycle and a typical 80% path stability is expected to be:

```
Synch time = adv rate per device  * #channels / (#advertisers * path stability * join duty cycle)
           = 0.16 s * 15 / (1 * 0.8 * 0.05)
           = 60 s
```

Setting a higher join duty cycle increases the power of the searching mote but allows it to synch up more quickly. If your network has a lot of 1-hop motes though, a low join duty cycle might not slow down the total join time by much at all. For applications with ultra-low power limits, the join duty cycle can be set as low as 0.5%.

Following synchronization, a WirelessHART device is required to wait for a random time longer than 30 s before sending in a join request. This is intended to reduce contention for limited joining resources in a large network.

3.2.1 Mote Joining

The join duty cycle can be changed using the *setParameter<JoinDutyCycle>* command. This command may be issued multiple times during the joining process but has no effect after join completion. A value different from the default will impact the synch time calculated above and the power at the mote during the search period.

After synchronizing and waiting for the WirelessHART random timeout, the joining mote sends in a *join request* consisting of power source and routing-capability information, as well as a list of heard neighbors. The manager responds with a run-time MAC layer authentication key, and a short address to be used by the mote in all communications from this point on. This message exchange is part of the security handshake that establishes encrypted communications between the manager or application, and mote, and is encrypted using a shared secret *join key*.In WirelessHART, there are two destinations upstream, the manager and the gateway. To each destination, the mote gets a route, a unicast session, and a broadcast session.

After the mote has progressed through the handshake, it transitions from using the links contained in advertisements to those explicitly added by the mananager. At this point it starts advertising for other new motes and may request additional bandwidth for publishing data. In this way the network will form 'inside out', starting with just the AP, then the one hop motes, then two hop motes, and so on. For any one mote to join, it need not ever be brought into range of the manager. It only needs to be within range of some motes that are in the network.

3.2.2 Discovery

In addition to the motes a joining mote hears, an ongoing *discovery* process runs continuously. Each discovery interval, each mote will randomly either listen (high probability) or transmit (low probability). This random process results in motes hearing most of their neighbors every 15 minutes. Motes tell the manager who they've heard in a periodic *health report,* which gives the manager a stream of potential path information to use in optimization and healing.

Mote ID

The 2-Byte mote ID is a nickname used in packets to avoid sending the full 8-byte MAC address over the air. The wireless network uses that nickname to save power and to maximize payload size for customer applications. The mote is given a mote ID by the Manager when it joins and the mote forgets it whenever it resets, as the Manager will assign it an ID the next time it joins. The mote ID is NOT guaranteed to remain unique to that mote over its lifetime - only the MAC address uniquely identifies that mote.

In SmartMesh Wireless HART, mote ID's are handed out in join order, and the mote ID is persisted through Manager reset or power cycle. However if a mote is deleted and rejoins, or is moved to another Manager, it will get a different mote ID. While it may be convenient or less confusing to use the nickname to identify a mote for a person analyzing the network at a particular time (e.g. using the Command Line Interface or CLI), Manager API's use the 8-byte MAC address and a software application should always ALWAYS use MAC address for mote interaction.

3.3 Bandwidth and Latency

SmartMesh WirelessHART total upstream network throughput is determined by how many packets/s can pass through an AP. Typical system throughput for a SmartMeshWirelessHART Manager (e.g. LTC5903-WHR) is ~24 packets/s . This bandwidth is shared between motes, so one mote could transmit at 24 packets/s, or 24 motes at 1 packet/s, or 10 motes at 2 packets/s and 1 mote at 4 packets/s, or any other equivalent combination. Each packet supports a 90 B payload, so, this is equivalent to ~2 kbps.

In WirelessHART, the manager tries to give each mote two upstream parents and a minimum of four upstream links. The two parents are required for a reliable mesh and the four upstream links are required to maintain time synchronization during difficult periods. In optimizing our networks, we test out links to new parents for some motes, so occasionally motes will have more than 2 parents. Depending on local and descendant traffic requirements, motes may end up with many more than 4 upstream links.

There are two ways of setting up bandwidth to carry data traffic. If all motes will generate data at the same interval, the base bandwidth method can be used. If some motes have more data generation than others, the services mode must be used.

In general, adding more links to motes:

- Decreases latency
- Increases packet/s throughput
- Increases power

There is no requirement to actually use all the bandwidth assigned to a mote. An application with low-latency requirements can request more bandwidth than it needs to get additional links for itself and its ancestors to decrease its upstream latency. Since the number of links required to meet a particular latency target varies by hop depth of a mote, we provide a mechanism for requesting a particular upstream latency. In this case, the manager assigns an appropriate number of links to try to get the mean latency to the target. Because of the uncertainty in path stability, we do not provide upper bounds on latency. For extreme cases, the low-latency pipe discussed below can be used. Typical system throughput is ~ 24 packets/s, shared between motes.

3.3.1 Base Bandwidth

The *requestedBasePkPeriod* parameter (default = 100,000 ms) is the interval between packets generated at all motes. The default setting is enough to carry all command and diagnostic packets. If all motes have the same data generation interval then this value can be set appropriately to provision the entire network. For example, if all motes are going to report every 10 seconds, setting *requestedBasePkPeriod* to 10,000 ms will give enough links to all motes to carry this traffic. This parameter is settable through API on the manager and applies to the entire network.

Using the base bandwidth method, motes can start reporting immediately after activation as long as backoff is implemented between the mote and the application.

3.3.2 Services

Applications wishing to support different data generation rates or run-time configurable data rates must use the services model, as must HART compliant products. In this model, the application is responsible for configuring the desired publish rate - the sensor processor could be pre-configured or an application message could be used to configure publishing dynamically - this is left to the OEM integrator. The *setParameter<service>* command allows the device to initiate a new service request or request an update to an existing service. The mote will then send a service request to the manager and the application must wait until a successful reply has been received before starting to publish data. After receiving a service request from a mote, the manager does not automatically send a reply when the required links have been added; the mote must continue polling the manager until it gets a reply affirming that the service is ready.

3.3.3 Cascading Links

If a mote has children, it has to add more links to carry the traffic of all its descendants. The calculation of this requirement is called *cascading* links. In SmartMesh WirelessHART, the manager doesn't just consider the input links to a device but instead calculates the expected traffic that will be on these links. In some cases, a device may have several upstream RX links that are used to maintain synch for its children but just a small number of upstream TX links because there isn't much traffic forwarded by this mote. All motes only get as many links as they need for network health and traffic in order to reduce power throughout the network.

3.3.4 Downstream Bandwidth

The WirelessHART AP has several *multicast* downstream TX links in addition to a single broadcast TX link. Only a subset of the 1-hop motes listen to each multicast link allowing them to maintain a low-power configuration while giving the AP more bandwidth into the network. This decreases the network formation time and increases the speed for applications which sequentially send downstream packets to each device in the network during steady-state operation.

There are two bandwidth profiles implemented for WirelessHART which differ only in their downstream capacity. The default profile is **P1** which allows about 4.5 pkt/s to be injected by the AP into the network. The ultra-low power profile **P2** has one-eighth of the downstream capacity. Using **P2** can save all devices about 9 µA but takes longer to build the network and greatly reduces downstream application bandwidth. As such, it is recommended only for deployments that have tight power budgets and do not require constant polling of devices by the application.

3.3.5 Fast Services on the Pipe

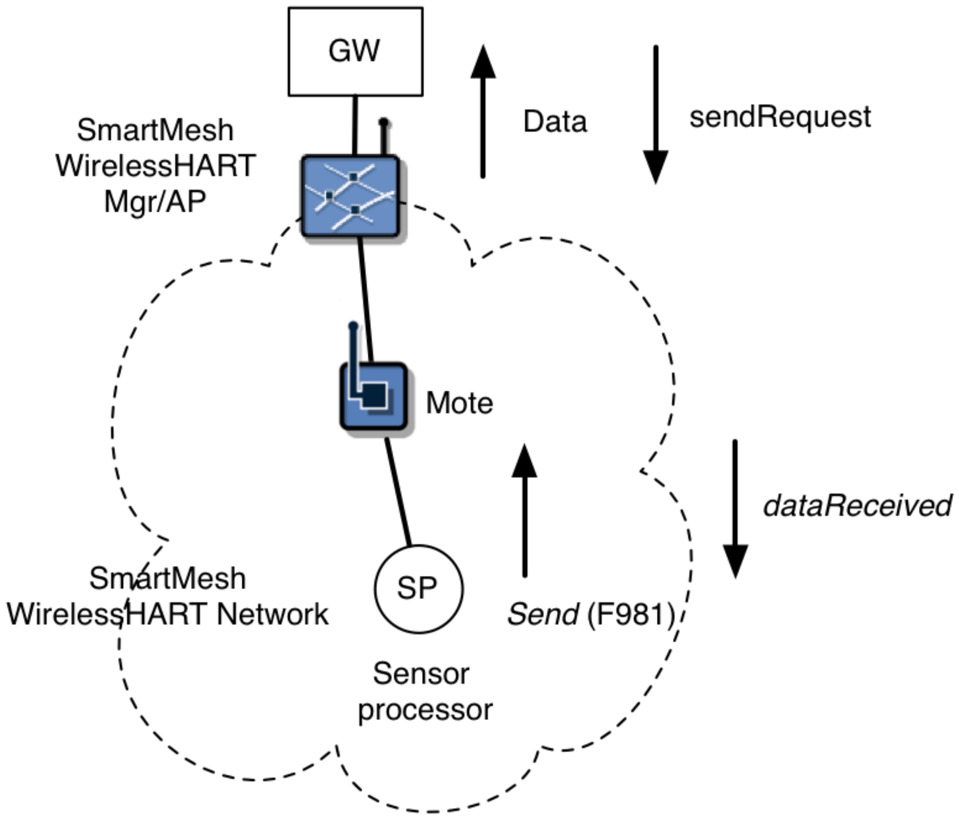
The WirelessHART manager is able to lay in fast upstream and/or downstream bandwidth on a special graph known as the *pipe*. The pipe links live on a short slotframe which allows the destination mote to publish and/or receive data quickly. The pipe can only be used for one destination mote at a time and is intended as a temporary addition to a network in order to rapidly upload or download a large amount of data.

The pipe should be explicitly requested by the application for a particular mote and it is expected to delete the pipe when it is no longer needed. If the mote is several hops deep, additional links are activated at each hop between the AP and the destination mote, so all of these devices will use extra current during time the pipe is in place. The route followed by the pipe is NOT a mesh, meaning that each hop points to a specific next mote. However, packets are allowed to use the regular upstream graph in addition to the upstream pipe, so there is no danger of a packet getting lost for a single path failure.

3.4 Data Traffic

An application connected to the manager can use the *sendRequest* API to send arbitrary packets to a mote. Messages must prepend a 4-byte header - 0x0000FC12 - this is to wrap the packet for the HART network layer, even for non-HART usage. Motes are addressed by EUI-64. When the API is called, a *callbackId* is returned - this *callbackId* will be included in the *netPacketSent* notification when this packet is injected into the network. When the packet is received at the mote, a *dataReceived* notification is generated which contains source address - Manager (0xF980) or Gateway (0xF981) and a sequence number in addition to the data payload.

The sensor processor may use the *send* API at the mote to send packets to the gateway or the manager. Certain fields that pertain to WirelessHART must be filled in (such as *appDomain*) regardless of whether the destination is a WirelessHART device. These will result in a *Data* notification, which contains the EUI-64 address of the mote, a timestamp, and the data payload.



3.5 Security

3.5.1 Security Layers

All packets in a SmartMesh network are authenticated on each layer, and encrypted end-to-end.

- Authentication - verifying that a message is from the stated sender, and that it has not been altered, or replayed
- Encryption - keeping payloads confidential

SmartMesh WirelessHART has several layers of security:

- Link-layer - packets are authenticated at each hop using a run-time key and a time-based counter - this ensures that only motes that are synchronized and been admitted into the network by the manager can send messages.
- End-to-end - packets are authenticated and encrypted end-to-end using run-time *session keys* and a shared counter - this ensures that only the intended recipient will understand the message (data privacy), and that replays, data corruption, or man-in-the-middle attacks can be avoided (data security).

When joining, motes send a *join request* to the network manager using a shared-secret *join key* known by the manager. Choice of keys is determined by the security mode, discussed below. The mote encrypts its join request with its join key, and the manager responds with a *session key* for end-to-end encryption of data traffic - this is known as the *security handshake*. Advertisements are a link-layer special case - they are authenticated with a well-known key to allow any new mote to authenticate them.

Once a mote has joined with the correct join key, it receives four session keys that are used to encrypt network data in operation:

- A mote-specific session key used for network management traffic
- A mote-specific session key used for application traffic
- A broadcast session key used by all motes for network management
- A broadcast session key used by all motes for application traffic

Using these four keys, all regular data publishes are encrypted by the generating mote and can only be decrypted at the manager. Neither eavesdroppers nor routing motes can decrypt the packet data. Similarly, responses to manager commands can only be read by the manager. Downstream commands to a particular mote cannot be understood by routing motes or unintended recipients. Only commands sent explicitly to all motes in the network can be understood by all, being decrypted using the appropriate broadcast key.

3.5.2 Security Modes

SmartMesh WirelessHART has a choice of security modes that determine how the manager decrypts the join request. The manager can do this in three different ways:

- Common Key: The least strict security mode is to accept a common join key. In this mode the manager will accept any mote that provides a join request encrypted with the common (network wide) join key. Dust Networks' network managers ship with a default common join key and Network ID that should be considered public knowledge. If the Network ID and common join key are left unchanged, overhearing and decrypting the packets that assign the session keys is difficult, however technically possible. Therefore, it is highly recommended to change the common join key to a secret one.
- Access Control List (ACL): The manager can also be set up to only accept motes on an access control list. The manager will take the join request, and first look for the serial number of the joining mote, and then decrypt the request with the associated join key. If both steps are successful, the mote will be accepted into the network. The ACL should be set up with a unique join key for every mote. This is the most secure mode, but requires the most effort on the part of the commissioning workforce, since it requires that the manager and all the motes be configured prior to deployment in order to work together. If these devices are already configured correctly, the installer need take no action - the mote will join when it hears an advertisement. The ACL can be set up with a common key - this provides some additional security over the common key alone, as the devices MAC address must be known to the manager for it to be able to join.
- Common Key -> ACL: It is also possible to form the network with a common key, then construct an ACL and assign unique keys to each mote over the air.

4 The SmartMesh WirelessHART Manager

4.1 Introduction

The SmartMesh WirelessHART Manager is the "brains" of a SmartMesh WirelessHART network. The manager is responsible for:

- Managing security information such as keys and nonce counters and distributing these to the network
- Determining the link schedule for every mote to ensure that time synchronization can be maintained and data service levels can be met
- Collecting health reports to continually update its picture of the network
- Responding to changes in topology
- Optimizing the network to minimize energy and spread traffic
- Presenting user interfaces to a network Host

The SmartMesh WirelessHART Manager provides configuration, management, and access point functionality for a network of SmartMesh WirelessHART motes. The embeddable manager includes a wireless transceiver, processor and memory, real-tim clock, embedded networking software, and interfaces to host systems. The SmartMesh WirelessHART Manager hosts an XML-RPC API that allows programmatic access to network control commands, performance statistics, and connectivity details. In addition, the manager offers administrative interfaces via its Web-based Admin Toolset utility and text-based Command Line Interface (CLI).

The LTP5903-WHR manager supports networks of up to 500 motes, and its AP is modularly certified.

4.1.1 Embeddable Manager

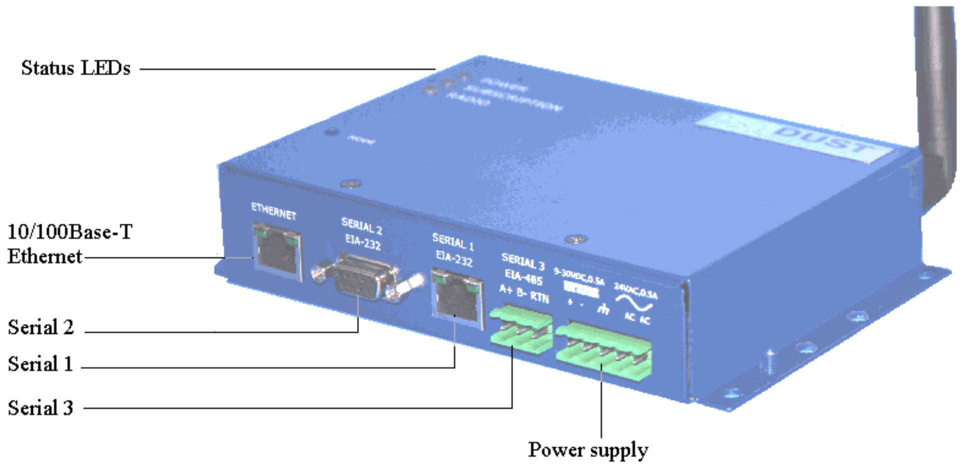
See the LTP5903-WHR Datasheet for details on the available ports, power supply considerations, signal timing, etc.

4.1.2 Packaged Manager

The LTP5903-WHR is also available as a fully integrated standalone manager. The LTP5903CEN-WHR manager provides a 10/100Base-T Ethernet interface and two serial interfaces (serial 3 is not enabled), a power supply connector, antenna, reset and factory restore switches, and status LEDs.

The status LEDs provide the following information:

- *Power (Green)* —The Power LED is on when the 12 V power supply provided with the LTP5903CEN-WHR is connected and functioning properly.
- *Subscription (Yellow)* —Indicates that a client program is subscribed to the manager API.
- *Radio (Yellow)* —Blinks when there is data activity over the radio.



Steps in Designing a Manager Client Application

Although the manager has many tasks it needs to do in order to manage a network, a client has relatively few. At a minimum, it should:

- Connect to the Manager
- Configure any parameters needed prior to join (such as *networkID*)
- Subscribe to notifications to observe mote status and collect data

The SmartMesh WirelessHART Manager API Guide covers other commands to configure the manager, e.g. configure security (use of ACL). The SmartMesh WirelessHART Manager CLI Guide guide covers using the human interface to observe manager activity (including traces of mote state or data).

Manager vs. Gateway

In a WirelessHART network, there is a distinction between a Manager and a Gateway:

- The Manager interacts with the wireless portion of the network, and builds and maintains the wireless mesh. There is an additional logical entity called the Security Manager which is responsible for key management and packet encryption/decryption that is also contained within our Manager. Often the manager is embedded within a Gateway box, but the two have distinct roles.
- The Gateway bridges between the customer application, e.g. an asset management system that is speaking wired HART, and the field devices in the wireless mesh. It does this through Manager APIs, including ones to send/receive HART and WirelessHART commands. The packaged manager can interface directly with a customer application (typically a non-HART customer), but isn't a WirelessHART gateway.

For more details, see HCF_Spec-085, section 6.2.

4.2 Manager Interfaces

4.2.1 Wired Interfaces

The physical inferfaces described are found on the LTP5903CEN-WHR standalone Manager. See the LTP5903-WHR Datasheet for details on connecting to these interfaces on an embeddable manager.

10/100Base-T Ethernet Interface

The 10/100Base-T Ethernet interface is a standard RJ45 connector which provides users with direct access to manager's XML-RPC API and the Admin Toolset utility. The Linux console and the Manager command line interface (CLI) can also be accessed via secure shell (SSH) over this interface. See the section "Managing Users and Passwords" in the SmartMesh WirelessHART Manager CLI Guide for login details.

Port	Description	Signaling
Ethernet	10/100Base-T Ethernet	IEEE 802.3 10/100Base-T (autosensing)

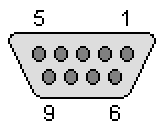
Ethernet Port Hardware Specifications

Serial 2 Interface

The Serial 2 interface (9-pin D-SUB female connector) is dedicated to the Manager's CLI.

Port	Description	Signaling
Serial 2	UART 9-pin	RS232 levels

Serial 2 Port Hardware Specifications
The Serial 2 port operates at 115200 baud, 8 data bits, No parity, 1 stop bit, no flow control.



9-pin D-SUB Female Connector

Serial 1 Interface

By default the Serial 1 interface is configured to expose the Manager's CLI, however iit requires special cabling to do so. See Assembling a 9-pin D-SUB Adapter for Serial 1.This interface can also be configured to run point-to-point protocol (PPP), allowing access the Manager’s XML-RPC API. See Configuring PPP on Serial 1 for details. Using PPP, an external gateway processor can also allow SSH into the Manager for access to the Linux console and the Manager CLI.

4.2.2 Software Interfaces

Command Line Interface (CLI)

The SmartMesh WirelessHART Manager provides a command line interface that can be accessed through a terminal program (such as TeraTerm or PuTTY). The CLI is intended for human interaction with the manager process, *e.g.* during development to observe various traces. System parameters such as Network ID can be configured through the CLI. System and network status information can also be retrieved via this interface. CLI access is protected with user name/password that may be changed by the user.

✓ For more detailed information on connection to the CLI and the commands that are available, refer to the SmartMesh WirelessHART Manager CLI Guide.

Application Programming Interface (API)

The SmartMesh WirelessHART Manager XML-RPC API provides a programmatic interface for interacting with the network. Host applications use the API to communicate with the manager. The XML-RPC API is an Extensible Markup Language (XML) interface that lets a client application send Remote Procedure Call (RPC) requests to the manager and receive responses and other data from the manager via XML-RPC. The API consists of a Control Channel and a Notification Channel. The Control Channel is used to establish connection and exchange commands and information about the SmartMesh Network. The Notification Channel is used to stream data and network events to the client Host program.

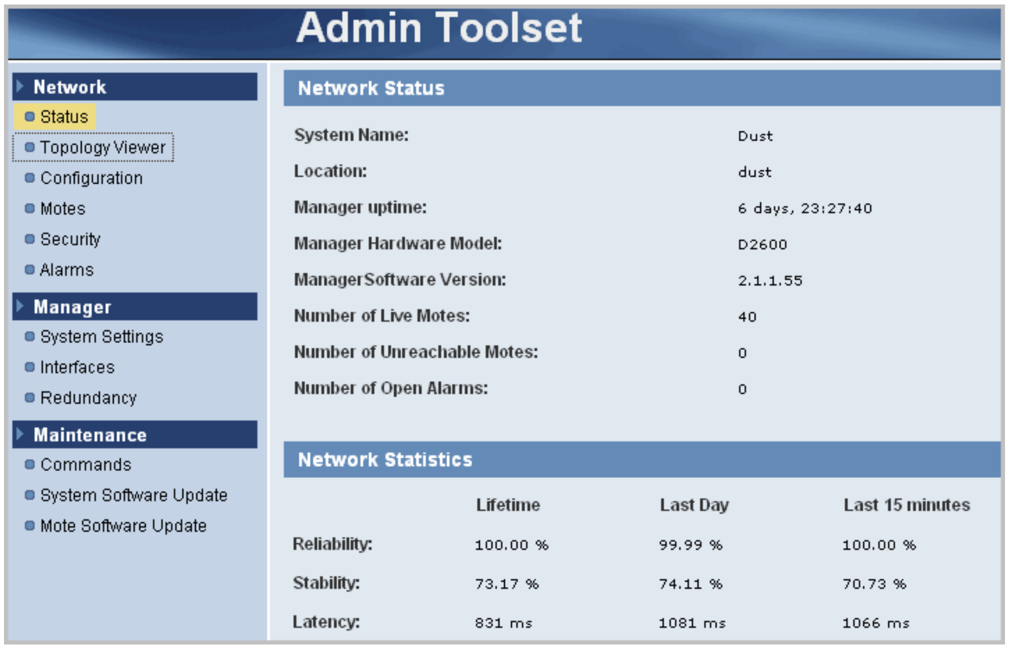
The following tools are available for experimenting with and communicating with the Manager API:

- The Python Developer SDK provides tools for performing common tasks and experimentation with the API.

✓ For more detailed information on connection to the API and the commands that are available, refer to the SmartMesh WirelessHART Manager API Guide.

Admin Toolset

The manager provides a Web-based administrative tool, called Admin Toolset, which can be used to view network statistics and mote and alarm information, configure serial and Ethernet port settings, set the clock or enable the Network Time Protocol (NTP) server, set the network security mode, and execute selective commands. The Admin Toolset provides a graphical view of the wireless network (Topology Viewer) and an interface for configuring the manager. You can also use Admin Toolset to upgrade the manager software as well as perform remote software updates on motes in the wireless network. The Admin Toolset utility is described in detail in the "Admin Toolset" section of the SmartMesh WirelessHART Tools Guide.



Network Status

System Name:	Dust
Location:	dust
Manager uptime:	6 days, 23:27:40
Manager Hardware Model:	D2600
ManagerSoftware Version:	2.1.1.55
Number of Live Motes:	40
Number of Unreachable Motes:	0
Number of Open Alarms:	0

Network Statistics

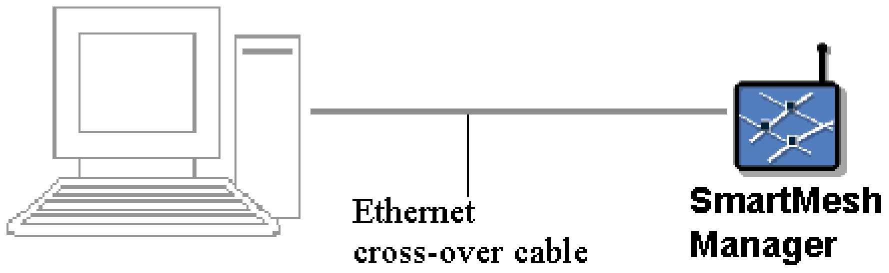
	Lifetime	Last Day	Last 15 minutes
Reliability:	100.00 %	99.99 %	100.00 %
Stability:	73.17 %	74.11 %	70.73 %
Latency:	831 ms	1081 ms	1066 ms

4.3 Connecting to the Manager

4.3.1 Connecting the Manager Directly to a Windows Computer

The SmartMesh WirelessHART Manager is pre-configured with the static IP address 192.168.99.100 for connection directly to a computer. You can temporarily set the computer IP address to a static address that enables the computer to communicate with the manager. The instructions below are for Windows XP. The steps will vary if you are using another OS on your computer.

⚠ To use the manager with its default static address, your computer will need to act as the Ethernet gateway. You may need to use an Ethernet cross-over cable to connect the manager to your computer. We recommend using the manager on a LAN using DHCP or an administrator supplied IP address, as described below.



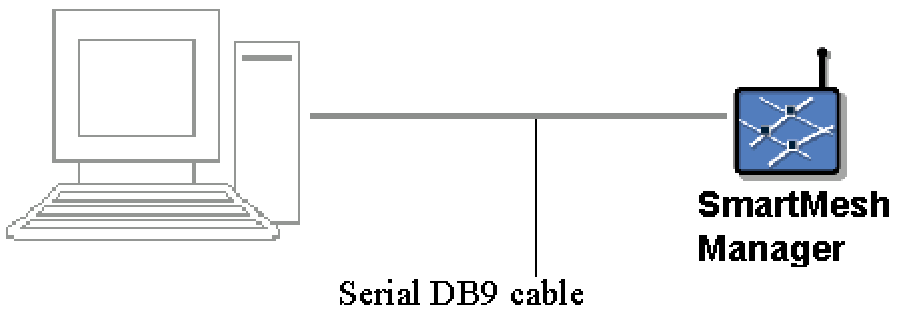
To set the Windows PC IP address to a static address:

1. On the **Start** menu, click **Control Panel**.
2. Double-click **Network Connections**.
3. Right-click **Local Area Connection**, and then click **Properties**.
4. Click **Internet Protocol (TCP/IP)**, and then click **Properties**.
5. Click **Use the following IP address**, and enter the following information:
   IP Address: `192.168.99.101`
   Subnet Mask: `255.255.255.0`
6. Click **OK** to close the dialog boxes.

When you are finished using the manager, you can switch your computer IP address back by selecting “Obtain an IP address automatically” on the General tab in the Internet Protocol (TCP/IP) Properties window.

4.3.2 Connecting to the Manager on the LAN

To connect to the manager on the LAN, you will need to change the IP address, as the default is not suitable for most networks. You can either configure the manager to use DHCP to obtain a LAN-assigned IP address, or assign a static LAN IP address to the manager. If you want to use a static LAN IP address, you will need to obtain this address from the LAN administrator.



To configure the Manager network settings:

1. Establish a console connection to the manager using HyperTerminal or similar software using the following serial settings: 115200 baud, 8 data bits, No parity, 1 stop bit, no flow control.
2. At the manager login, enter the user name (default: `dust`)
3. At the manager password, enter password (default: `dust`)
4. Connect the manager to the LAN using the Ethernet straight-through cable.
5. Configure the manager to use DHCP or assign a static LAN IP address.

To configure the manager to use DHCP, use the following commands on the Linux prompt:

```
dust@manager:~$ sudo ifswitch-to-dhcp
Switching interface to DHCP... eth0 down interfaces modified setting ethernet options: speed=100
duplex=full
ADDRCONF(NETDEV_UP): eth0: link is not ready
udhcpc (v1.13.2) started
Sending discover...
Sending discover...
Sending discover...
No lease, forking to background
eth0 up Done!
```

⚠ The manager must be connected to the LAN (step 4) before the `sudo ifswitch-to-dhcp` command is issued or an IP address will not be assigned to the manager by the DHCP server.

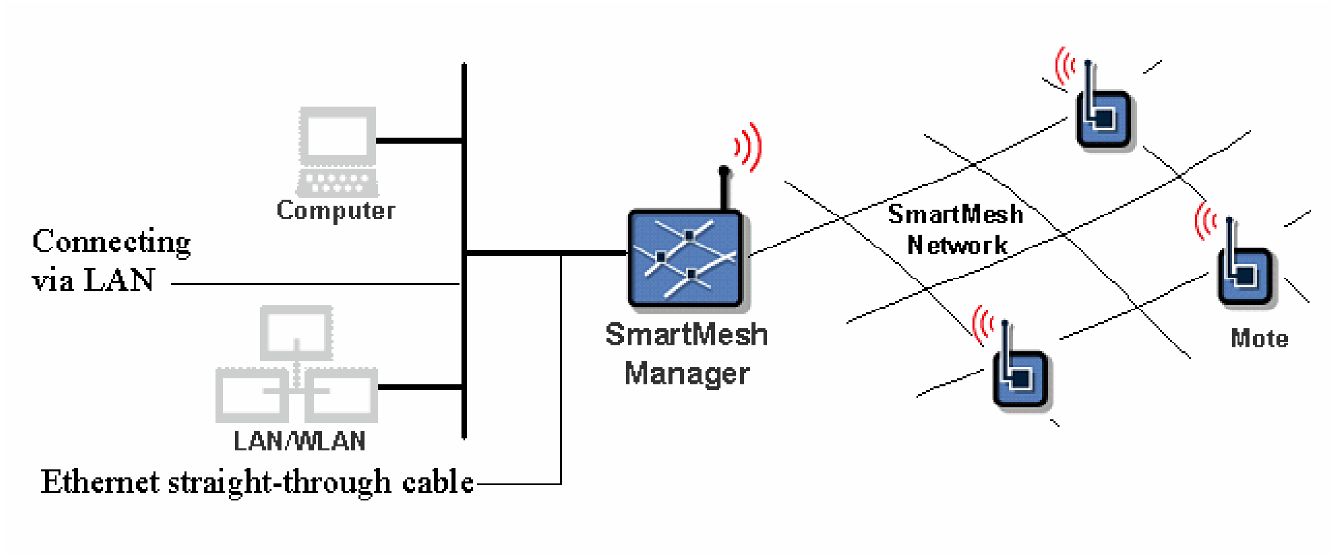
To configure the manager to use a static LAN IP address (here 172.16.1.103), enter the following on the Linux command prompt:

```
dust@manager:~$ sudo ifswitch-to-static 172.16.1.103
Switching interface to static IP allocation... ifdown: interface eth0 not configured
eth0 down interfaces modified setting ethernet options: speed=100 duplex=full
ADDRCONF(NETDEV_UP): eth0: link is not ready
eth0 up Done!
```

To verify that the manager's IP address has been changed (on eth0), enter:

```
dust@manager:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:17:0D:80:10:5B
          inet addr:172.16.1.109  Bcast:172.16.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:1 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:90 (90.0 B)
          Interrupt:21 Base address:0x4000
...
```

If the manager is configured to use DHCP, it must always have an Ethernet connection to the LAN when it is powered on or reset or it will not receive a LAN IP address from the DHCP server. If you power on or reset the manager before connecting it to the LAN, you will need to establish a terminal connection to the manager and issue the `sudo ifswitch-to-dhcp` command to prompt the DHCP server to assign the manager a LAN IP address.



4.3.3 Accessing Admin Toolset

The SmartMesh WirelessHART Manager can be administered using the Admin Toolset web interface. To access Admin Toolset:

1. Determine the manager's IP address
2. In your web browser, open a connection to `https://manager's_ip_address/` - note that this is a secure connection (https)
3. Admin Toolset uses a self-signed certificate; most web browsers will warn about this and you will need to click through a **Proceed Anyway** button to continue
4. Admin Toolset requires authentication; the default username is `system` and password is `system`

4.3.4 Configuring the Firewall

Beginning with SmartMesh WirelessHART Manager manager version 4.0, the firewall installed on the manager allows you to restrict communication on some of the manager ports. The table below lists the ports that are open by default when the manager is shipped. These ports can be restricted by configuring the file `/etc/firewall.conf`. Communication can be restricted to a specific range of IP addresses, or to `localhost` (effectively blocking the port).

Port #	Type	Service
22	TCP	SSH
443	TCP	HTTPS
53	UDP	DNS
123	TCP/UDP	NTP
4445	TCP	XML API, plaintext
4446	TCP	XML API, SSL
24112	TCP	XML API notification, plaintext
24113	TCP	XML notification, SSL

Manager Ports
The ports for the XML API can be configured in the `system.ini` file. If the XML API ports are changed, the firewall configuration must be adjusted to match.

To configure the XML API:

1. Edit the contents of `/opt/dust-manager/conf/config/system.ini`. The *PORT_CTRL_TCP* and *PORT_CTRL_SSL* parameters control the plaintext and SSL API control ports. The *PORT_NOTIF_TCP* and *PORT_NOTIF_SSL* control the plaintext and SSL notification ports.
   The parameters must be uncommented (remove the # at the start of the line) to take effect.
2. Restart the Manager software.

To configure the firewall:

1. Access the Linux login prompt by entering the CLI username and password (default is *`dust`* / *`dust`*).
2. Edit the contents of `/etc/firewall.conf` by running:

   ```
   $ sudo vi /etc/firewall.conf
   ```

   See the instructions in the file for an example of how the entries can be changed.

4.3.5 Changing the CLI Password

You can change the default Linux login password. To change the password, first login with your old name/password and follow these steps:

1. Enter: *passwd*
2. When prompted, enter the old password.
3. Enter the new password.
4. Re-enter the new password.

4.4 Administering the Manager

4.4.1 Root Access

In addition to the web-based Admin Toolset which is described in the SmartMesh WirelessHART Tools Guide, the SmartMesh WirelessHART Manager can be administered through the Linux console from an SSH connection or a serial connection to the Serial 2 port. See the SmartMesh WirelessHART Manager CLI Guide for connection details. The `dust` user has `sudo` access to some system administration commands. Although root access is not normally required, if needed the user can change to root via the `su` command:

```
$ su sur+cycl3s
```

The root password should be considered public knowledge. Users cannot log in remotely as *root* - they must first login as *dust*, and then change users to *root* via the Linux `su` command. For that reason it is critical that if the manager is addressable from the outside world (i.e. it is not behind a firewall), then the *dust* password be changed. Changing the *dust* password is sufficient for most attack scenarios. This can be done with the following script (e.g. to change it 'newPassword!'):

```
$ sudo set-dust-password newPassword!
```

4.4.2 Network ID

Overlapping installations can be arranged into separate networks by assigning unique Network IDs to the motes and Manager in each network. Motes won't communicate with motes or a Manager that are using a different Network ID. The Network ID stored in the Manager configuration can be updated through the API *setConfig* or the CLI `set network` command. The Manager must be restarted to use the new Network ID.

Each SmartMesh Manager ships with a default Network ID. To change the Network ID, use the commands described in the following table. The *exchangeNetworkId* command can be used to change the Network ID for all motes that are in a particular network. The *exchangeNetworkId* command reliably pushes a new Network ID to all the motes in a network. If a mote fails to acknowledge the exchange Network ID operation, the mote is removed from the network. The *exchangeNetworkId* operation may take several minutes to complete. The network ID can also be changed on a mote-by-mote basis using the *exchangeMoteNetworkId* command.

Command	**Description**
Use *setConfig* to set the *networkId* field of the *Network* element	Changes Network ID on the manager.

exchangeNetworkId	Changes Network ID on the manager and all network motes.
exchangeMoteNetworkId	Changes the Network ID for a specified mote.

Exchange Network ID command

The *exchangeNetworkId* command initiates an update of the Network ID for all motes in the network. The Network ID is exchanged without loss of data. The execution of the exchange is set in the future to allow time for the command to propagate across the network and allow time for re-transmissions. The manager generates a *sysCmdFinish* event when the command is synchronously executed by the motes. The delay is typically tens of minutes after the *exchangeNetworkId* command is issued and depends on the network size and configuration. For networks using the **P1** configuration, the delay is ~21 minutes. During this period, no additional *exchangeNetworkId* commands may be issued. After the exchange is completed, the manager and motes must be reset for the new Network ID to become effective.

Note that network motes must be in the Operational state when the *exchangeNetworkId* command is issued in order for them to receive the new Network ID. For best results, use the following procedure:

1. Before exchanging the Network ID, first determine if all network motes are operational by using the *getConfig* command to retrieve the state of all motes in the network. If there are non-operational motes, wait for them to rejoin the network. If they do not rejoin within an hour, troubleshoot the problem (for example, check the mote batteries).
2. When all motes are operational, issue the *exchangeNetworkId* command.
3. When the *sysCmdFinish* event is received, reissue the *getConfig* command to determine if all motes are operational. Do one of the following, depending on the outcome:
    1. If all motes are operational, they have all received the new Network ID. Reset the network.
    2. If some motes are non-operational, wait up to an hour to see if the missing motes can rejoin the network. If the missing motes do not rejoin, repeat steps 2 and 3 with the old Network ID so that these motes can rejoin. Then repeat steps 1-3 with the new Network ID.

4.4.3 Network Time

The Manager handles time synchronization in the network. By default, the Manager is configured to automatically start the network when it boots. Once the network starts, the time in the network (ASN) will drift synchronously on all in-network devices. The manager will use Linux system time as a reference, pushing the mapping between UTC and ASN periodically to the network.

A Host Application can query the current time on the Manager using the getTime API.

The response contains the current Manager time in several formats:

- **Unix (UTC) time: seconds and microseconds since Jan 1, 1970 in UTC**
- **ASN**: Absolute Slot Number, i.e. number of slots since ASN=0, which maps to the manager start time.

4.4.4 Software Licensing

The SmartMesh WirelessHART Manager supports a software license mechanism that allows optional software features to be enabled in the field. For example, a manager may be enabled for large network size with the purchase of a software license key.

To exercise the software license mechanism:

1. Obtain the following information specific to your manager. This information is needed to obtain an updated software license key.
    1. Retrieve the manager's Serial Number by using the `get system` command. This information is also listed in the System element in the Manager API.
    2. Retrieve the current software license value by using the `exec getLicense` command from the Manager CLI. The *getLicense* command is also available in the Manager API.
2. Contact Dust Networks with the information retrieved in step 1 to obtain an updated software license key.
3. Use the `exec setLicence` command to enter the new license key. The *setLicense* command is also available in the Manager API.
4. Restart the manager software to enable the new license. Once the new license is enabled, it will possible to enable additional features through the Manager configuration. Some features may require the Manager to be restarted after the configuration is updated.

4.5 Network Activity

4.5.1 Network Structure and Formation

The SmartMesh WirelessHART Manager performs automatic network management operations to maintain the health of the network as well as optimize it for the lowest power consumption and highest reliability. The manager also dynamically makes changes to the mesh as conditions in the network vary, such as path stability changes due to interference, addition and removal of service requirements, and addition and removal of devices. The customer applications never have to get involved in the management aspects of the network.

If motes are powered up in the vicinity of a manager, they will start joining almost immediately. The AP advertises on average twice per second and motes, once joined, advertise once every two seconds. All motes will advertise except for those explicitly designated "non-routing". The manager does not deactivate advertising; if desired, advertising must be deactivated by the application and can save power on each mote. Such an application must be able to detect lost motes and reactivate advertisement to retrieve them.

In order for a mote to join a Manager's network, the mote must be assigned the Network ID used by the Manager. The Manager and mote exchange several messages in the join process to establish security keys and proper network routing. The mote goes through several state changes during the join process, ending in the **Operational** state. The Manager will detect if a mote leaves the network and will update the mote's state to **Lost**.

An External application can keep track of motes joining and leaving the network by listening to Event notifications.

4.5.2 Communicating with Motes

Once motes are **Operational**, the customer's host application can communicating with motes and with the attached sensor processors via Manager APIs. The application uses the *sendRequest* API to send packets to a mote and receives upstream data from motes by listening to *data* notifications.

4.5.3 Network Health

The motes and Manager each keep track of network and device statistics.

Statistics queries are available through the following API commands:

- *getConfig<network>* provides network-wide statistics.
- *getConfig<motes>* provides statistics accumulated from communicating with a mote.

From the CLI, the various `show` commands will return information about the network

4.5.4 Health Reports

Motes periodically report statistics to the Manager in health reports. The Manager uses the health reports to determine a mote's neighbors and path stability for network optimization. The Manager accumulates interval and lifetime statistics but the Manager does not store individual health reports.

Health reports are available as notifications in the API (as of version 4.1), so an external application connected to the Manager can subscribe to health report notifications and track network statistics in detail over time.

4.5.5 Optimization

Based on information in the health reports, the manager continuously works on improving the network. For each mote, the manager considers how the existing parents compare to other neighbors discovered by the mote. If one existing parent looks better than the other, or if a discovered but unused neighbor looks like it would make a better parent, the manager will add a link from the mote to the better parent. This is called the *optimization add* cycle. Comparing the parents involves comparing a scoring function that takes into account path stability, the hop distance to the AP, and the relative number of links on the mote. The manager tries to reduce the energy consumption in the network by having each packet transit fewer, higher stability hops and tries to balance the load at the busiest motes. During the add cycle, the manager often has to guess at which paths are going to be best for the network and tries them out by adding single links to each mote.

After an hour, the manager has collected four health reports from the mote with the new parent and then has sufficient data to quantify which parents are best. Any mote with an extra link at this stage will delete the link to its worst parent, again by the same scoring metric. This is called the *optimization delete* cycle. The process continues for the life of the network, alternating add-delete-add-delete, each one hour apart. Running continuously allows the network to adapt to any changes in the environment.

4.6 Network Bandwidth Control

The SmartMesh WirelessHART Manager supports dynamic bandwidth to accommodate the bandwidth requirements of complex applications common in industrial monitoring and control. For example, a SmartMesh WirelessHART network can support request/response maintenance traffic from a controller or gateway down to a network device while simultaneously activating a fast pipe for block transfer at the request of a worker in the field. The manager precisely allocates network resources to support the bandwidth requirements of the application while maintaining ultra-low power consumption across the network. To support these applications, bandwidth can be requested by either the network manager using the manager API commands or requested by network devices through bandwidth service requests. In response to these bandwidth controls, the network manager will schedule links in the network to increase bandwidth, or in some cases may activate a fast bandwidth pipe.

This section describes the manager API commands for both requesting bandwidth and controlling the bandwidth allocated to service requests. The manager API enables the following:

- Limits on Service-Requested bandwidthdefine maximum bandwidth that may be granted to service requests.
- Manager-requested bandwidthdefines network-wide minimum bandwidth.
- Allocated bandwidthreturn the total bandwidth allocated to a specific device.
- Pipe activationenables the fast pipe to be activated or deactivated through the manager API and provides visibility to the pipe status.

Since fast pipes may be activated by the network manager as a result of a manager API command or from a service request from a network device, it is worth noting that the device that requested the bandwidth owns and controls it. That is, a pipe activated by service request can only be deactivated by a subsequent request from the same device. Similarly, a pipe activated by manager API can only be deactivated by manager API. In addition to manager-initiated bandwidth control, there are also mechanisms by which the mote can exercise bandwidth control. For more information, see the “Bandwidth Services” section in the SmartMesh WirelessHART Mote API Guide.

⚠ SmartMesh motes ask for bandwidth in terms of the interval between packets. For example, a mote expected to generate one packet every 30s requests a 30s service. A mote requesting a 5s service will need more links per second to carry this traffic, and hence needs more bandwidth. Because of this relationship, the minimum packet interval allowed corresponds to the maximum bandwidth.

Manager API Parameter	**Description (see SmartMesh WirelessHART Manager API Guide)**
User-Settable Bandwidth Limits	

minServicesPkPeriod	Limits non-pipe services. Defines minimum data interval (in ms/packet) that a single mote may be allocated for the total of non-pipe, user requested bandwidth. Limits service requests from a mote. See the *Network* element
minPipePkPeriod	Limits pipe bandwidth. Limits bandwidth on all pipes (both manager-API requested and pipes as a result of service request). Most useful for regulating service requests from field devices. See the *Network* element
Manager-API Driven Bandwidth Requests	
requestedBasePkPeriod	Network-wide minimum data interval (in ms/packet). Note that the *requestedBasePkPeriod* parameter applies to manager-controlled bandwidth, which is independent from bandwidth requested through mote service requests. Any mote service requests will be on top of *requestedBasePkPeriod*. See the *Network* element
Allocated Bandwidth	
allocatedPipePkPeriod	Pipe bandwidth. Returns usable bandwidth allocated to the pipe. See the *Mote* element
netServiceDenied	XML API notification that indicates when a network service has been denied. The *minServicesPkPeriod* limit has been reached or there is a bottleneck in the network, as indicated by the Mote's *needNeighbor* flag. See the *netServiceDenied* notification
Pipe Activation	
activateFastPipe command	XML API command to turn on a pipe. This command requests activation of a fast pipe to a specific Mote. The pipe can be upstream, downstream or bidirectional. See the *activateFastPipe* command
netPipeOn	XML API pipe activation notification. This notification is sent by the manager API, signifying that a pipe has been activated. The notification contains the allocated bandwidth for the pipe. See the *netPipeOn* notification
pipeStatus	The status of a mote's pipe is indicated in the Mote's *pipeStatus* parameter. See the *Mote* element

4.7 Access Control

4.7.1 Network Security

The manager API offers a choice of network security modes:

- Accept Common Join Key
- Accept ACL
- Quarantine on Common Join Key (Added in Manager 4.1)

The security mode is specified as part of the Security element.

Common Join Key Mode

If the security mode is set to *Accept Common Join Key*, any mote sharing the network manager's common join key is allowed into the network, as well as any mote on the access control list (see the following section, ACL Mode). The join key is a symmetric encryption key that is shared between the manager and the motes in its network. The manager ships with a default common join key, which should be considered public knowledge. The default common join key is typically changed in advance of network deployment, but can also be changed after the network has formed. If the common join key is left unchanged, overhearing and decrypting the packets that assign the session keys is difficult, but technically possible. Therefore, it is highly recommended to change the common join key to a secret one.

To change the common join key, use the commands described in the following table.

Command	Description
Use *setConfig* to set the *commonJoinKey* field of the *Security* element	Changes the common join key on the manager.
exchangeJoinKey	Changes the common join key on the manager and network motes.

ACL Mode

If the security mode is set to *Accept ACL*, only motes on the manager's Access Control List (ACL) are allowed to join the network. If the ACL contains no entries, no motes will be allowed to join the network. The ACL is managed by the OEM host application, which uses the *createConfig* command to add motes and the *deleteConfig* command to remove motes from the ACL. Note that the ACL list can be set up with the same join key for all motes, or a unique join key for each mote.

Using the ACL with a unique join key for each mote provides the highest security but requires the most effort on the part of the commissioning workforce to configure the manager and the motes to work together. The ACL can be configured in advance of network deployment if the MAC address of each network mote is known. If the MAC addresses are not known, the motes may be allowed to join the network using the default common join key and the ACL can be configured after getting the mote MAC addresses from the *netMoteJoin* event notifications that are sent to the manager when a mote joins the network. Before adding a new mote to an ACL network, you must first add the mote's MAC address and join key to the ACL list. When removing a mote from an ACL network, you should remove the mote MAC address from the ACL to preserve network security.

To change the mote join key in Accept ACL mode, use the commands described in the following table.

After changing the security mode to accept ACL it is advised that you reset the network to remove any motes that may have previously joined the network using the common join key.

Command	**Description**
Use *setConfig* to set the *joinKey* field of a mote listed in the ACL.	Changes a mote join key in the manager's ACL.
exchangeMoteJoinKey	Changes a mote join key on the manager and the specified mote.

Quarantine Mode

(Added in Manager 4.1)

If the security mode is set to *quarantineOnCommonJoinKey*, the motes that join with the common join key are allowed into the network, but placed in quarantine.

- Motes are configured with a common Quarantine join key at the factory.
- The manager is configured in the *quarantineOnCommonJoinKey* security mode.
- When a mote attempts to join with the quarantine key, it is placed in quarantine, and a new event notification *netMoteJoinQuarantine* is generated. This has the same fields as the existing *netMoteJoin* notification.
- The mote is advanced to a new *Quarantine* state, equivalent to the *Connected* state.
- The GW uses the *exchangeMoteJoinKey* API to change to a unique (or common non-quarantine) key on the mote and add the mote to the ACL. The ACL is changed only after the mote has responded to the reliable command. See note below.
- The GW uses the new *promoteToOperational* API to instruct the manager to finish the join process.

- The manager advances the mote to the *operational* state.
- When a mote joins with a unique (or shared non-quarantine) key, if it is on the ACL it is allowed, otherwise it is denied. If a mote that is on the ACL attempts to join with the quarantine key, it is denied.

Note: In rare cases, the mote can receive the join key change, but the manager does not receive its reply – this will result in the mote being reset, and subsequent join attempts will be rejected due to the mote using the wrong key.

Updating a Mote's Join Key

The *exchangeMoteJoinKey* command is used to update both the manager's ACL and the join key stored on a mote. The *exchangeMoteJoinKey* command sends a command to the mote to set the new join key. When the Manager receives a successful response from the mote, it updates the ACL entry for that mote.

The join key is a symmetric encryption key that is used by the manager and motes to encrypt and decrypt the join messages that they exchange when the mote is attempting to join the network. If the ACL does not contain an entry with this MAC address, the manager creates a new entry. When the exchange is completed, a *sysCmdFinish* event is generated. The join key is exchanged without loss of data, and the mote does not need to be reset.

⚠ The time required to complete the command and issue a *sysCmdFinish* notification may vary depending on the size and type of the network. During this period no additional *exchangeMoteJoinKey* commands may be issued.

Note that mote whose join key you wish to change must be in the Operational state when the *exchangeMoteJoinKey* command is issued in order to receive the new join key. For best results, use the following procedure:

1. Before exchanging the join key, first determine if the mote is operational by using the *getConfig* command to retrieve the state of all motes in the network. If the mote is non-operational, wait for it to rejoin the network. If it does not rejoin within an hour, troubleshoot the problem (for example, check the mote batteries).
2. When the mote is operational, issue the *exchangeMoteJoinKey* command.
3. When the *sysCmdFinish* event is received, reissue the *getConfig* command to determine if the mote is still operational. Do one of the following depending on the outcome:
    1. If the mote is operational, it has received the new join key and no further action is needed.
    2. If the mote is non-operational, wait up to an hour to see if it can rejoin the network. If it does not rejoin, manually update the manager's ACL with the old join key and wait for the mote to rejoin. Then repeat steps 1-3 with the new join key.

4.7.2 Limiting the Maximum Number of Motes

The *maxMotes* parameter in the *Network* element defines the maximum number of motes expected in the network. This parameter can be used to prevent additional motes from joining the network. The manager keeps track of all motes that have joined the network since the last restart of the manager software. Once the *maxMotes* limit is reached, no additional motes are allowed to join the network. If a mote joined the network, then was removed, it is still included in the number of motes count unless it is explicitly deleted with the *deleteConfig* command.

⚠ The Access Point (AP) mote is counted in the total number of motes, so to allow *n* motes to join the network, *maxMotes* must be set to *n+1*.

4.8 Over-the-Air-Programming

4.8.1 Background

WirelessHART motes can have their firmware updated remotely via Over-the-Air-Programming (OTAP). This process can be initiated in a number of ways (via Manager CLI, API, or via the Admin Toolset Utility) but in general consists of the following steps:

1. Updated firmware is provided by Linear in the form of an `.otap` file for DN2510-based devices, or an `.otap2` file for LTC5800-based devices (requires Manager version >= 4.1).
2. The software is uploaded to the Manager, and a command is invoked to start the OTAP process.
3. The manager performs a handshake with all motes to determine which motes can accept this update (the receive list), and prepares them for update.
4. The OTAP file is divided into blocks and the Manager sends it to all devices - only devices on the receive list will do anything with the file.
5. The Manager queries all motes on the receive list to verify that the image was received and is valid. Missed blocks are repeated until all eligible motes have received the file.
6. The Manager sends the commit message to all motes that received the file. This will cause them to reprogram their flash with the new firmware.
7. Motes are reset and rejoin the network using the new firmware.

In some cases, the OEM may not wish to allow mote updates without explicit application authorization. The setNVParameter<OTAPlockout> mote API can be used to enable/disable a mote participating in OTAP updates. By default, the mote accepts OTAP upgrades.

OTAP is a slow process - this is so that mote average current isn't markedly increased by the additional traffic and flash writing. It will typically take 5+ hours to complete an upgrade. Networks that are larger and less dense (100's of nodes) and those with poorer path stability (< 80%) can increase the time by more than 2x.

4.8.2 OTAP through a Serial Port

To OTAP through a serial port:

1. Use a SCP (e.g. WinSCP) application on the PC to copy the OTAP files to the manager:
    1. OTAP files go in the `/opt/dust-manager/otap`. You will need to create this directory if it does not already exist.
    2. Place both AP and Mote this directory. The OTAP function will recognize via the headers which file is which.
2. Start the OTAP process
    1. Log into the manager CLI (nwconsole). Login directions can be found in the SmartMesh WirelessHART Manager CLI Guide.
    2. Enter

        ```
        otap start -n 25
        ```

        This will start OTAP with 25 rounds of retries. Retries are necessary since typically a small number of OTAP packets (which are sent best effort) will not reach all motes in a single round.
3. Periodically check to see if OTAP has completed:

    ```
    1. otap status
    ```

4. If all passed and all motes return "COMMIT":
    1. Reset the AP

        ```
        exec reset mote 1
        ```

5. If OTAP does not successfully compete:
    1. Cancel and consult the troubleshooting section

        ```
        otap stop
        ```

4.8.3 OTAP Using Admin Toolset

See the SmartMesh WirelessHART Tools Guide for details on performing software updates through Admin Toolset.

4.8.4 OTAP via Manager API

Once the .otap file has been placed in the /root/otap directory (see above), OTAP can be initiated using the *startOtap <numRetries>* command on the XML or serial manager APIs.

OTAP can be canceled any time before devices are sent the commit message using the *cancelOTAP* command on the XML or serial manager APIs. If it is too late to cancel, an error (-561) will be returned on the XML API only - there is no feedback for canceling OTAP on the serial API.

Details about the OTAP process in progress can be found in the *config/Network/OtapStatus* schema element of the XML API.

4.8.5 Troubleshooting an Unsuccessful OTAP

- Because best effort is used, in rare instances of poor path stability 25 rounds of retries is not sufficient to OTAP all motes. If OTAP fails, try increasing the retry count.
- In mixed networks (Some Eterna-based motes, some DN2510-based motes), each type of device must OTAP'd separately.

4.9 Restoring Manager Factory Default Settings

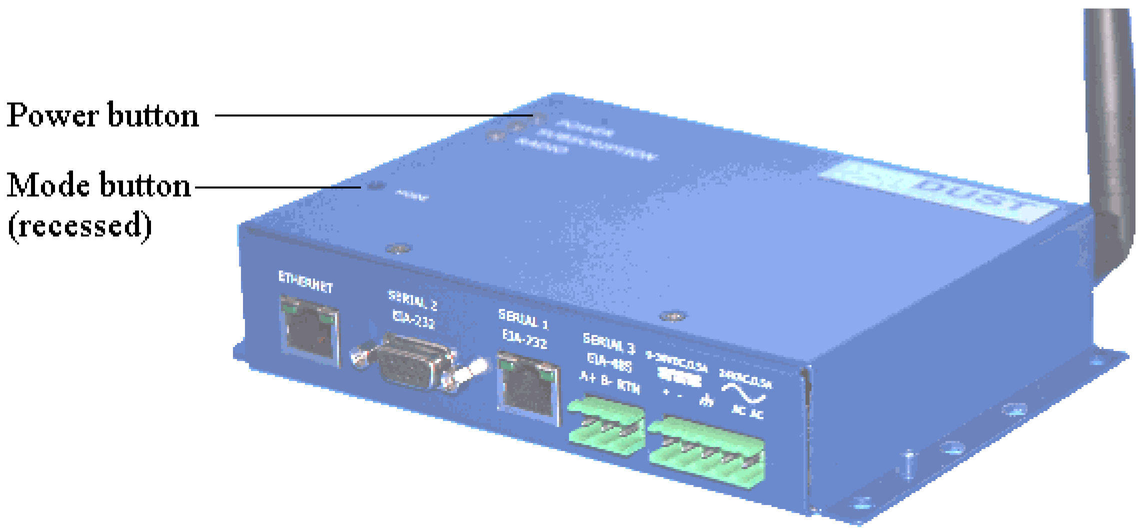
Use the following procedure if you need to restore the factory default settings to the manager.

The following factory default settings are restored:

- IP address
- PPP settings
- Serial port settings
- User name and password
- Wireless network and mote configuration settings
- Wireless network ID and common join key
- Access control list (cleared)
- Log files (cleared)
- Mote list (cleared)

Using the restore button (LTP5903CEN-WHR):

- Press and hold the restore button (labeled "Mode*")* button while you press and release the *Reset*(*Power)* button. Continue holding the Mode button down for another 20 seconds, and then release. Both of these buttons are recessed, so you will need a tool such as a large paper clip to press them.



From the Linux Prompt (applies to LTP5903-WHR and LTP5903CEN-WHR):

From the Linux command line (logged in as user `dust`), execute this command:

```
sudo /usr/bin/restore-factory-conf
```

From the Admin Toolset web interface:

Go to **Maintenance -> Commands -> Restore Factory Configuration** and click on the **Restore** button.

4.10 Channel Blacklisting

Although the network may operate on as few as five channels, it is recommended that the network run on as many channels as possible for greater resiliency and more overall bandwidth.

The default behavior for SmartMesh Networks is to blacklist only the sixteenth channel (2480 MHz) to comply with requirements in the United States (as regulated by FCC) and Canada (as regulated by IC). An odd number of channels must be blacklisted, but you are free to use the *setConfig* command to choose which ones. You are responsible for ensuring that the allowed frequencies conform with local RF regulations.

The following channel frequencies may be specified:

```
2405, 2410, 2415, 2420, 2425, 2430, 2435, 2440, 2445, 2450, 2455, 2460, 2465, 2470, 2475, 2480
```

Channel 0 (2405 MHz) corresponds to IEEE channel 11, and Channel 15 (2480 MHz) corresponds to IEEE channel 26.

Note that Dust modules (including the AP) are certified as Frequency Hopping devices - if you operate any device in the network at above +10 dBm EIRP, you may not be able to use blacklisting in some jurisdictions.

Where blacklisting is permitted, the number of blacklisted channels must be an odd number. An error is generated when the number of blacklisted channels is set to an even number. Starting with Manager version 4.1.3, if more than 9 channels remain, the network will operate in EN 300 328 rev. 1.8.1 compliant mode. If 5 or 7 channels remain, the network will not be EN 300 328 compliant.

4.11 INI Files

Many aspects of manager (aka *controller*) low-level behavior can be configured through the following set of `.ini` files:

4.11.1 system.ini

`$SMARTMESH_HOME/conf/config/system.ini`

The `system.ini` file contains settings that are specific to the configuration of particular system (box). These settings are designed to be changed by system administrators at install/configuration time.

4.11.2 platform.ini

`$SMARTMESH_HOME/conf/config/platform.ini`

The `platform.ini` file contains settings that are specific to platform on which the manager runs. These settings are intended to be modified by Dust or the OEM.

4.11.3 dcc.ini

`$SMARTMESH_HOME/conf/config/dcc.ini`

The `dcc.ini` file includes internal manager settings that are morally not customer-facing. This file does not exist in a default installation. It can be created by the `dust` user in order to override specific variables.

4.11.4 Modifying ini variables

Use the `set-conf-param` script to modify ini variables:

```
set-conf-param <file> <parameter> [value]
```

This script only examines the existing configuration file. It does not know about default values compiled into the software. Therefore, if a file does not exist or a variable is not present in the configuration file, the script will not return a value.

For example:

```
dust@manager$ set-conf-param $SMARTMESH_HOME/conf/config/dcc.ini APD_APM_MAX_NUM_TRIES = 5
```

4.11.5 Default dcc.ini

The `dcc.ini` file does not exist by default. The parameters contained in the `dcc.ini` file modify the way the manager builds, maintains, and optimizes the network. Most settings have not been extensively tested if they have been tested at all. Changing `dcc.ini` can result in unpredictable behavior, or a network that does not function. Please contact Applications Engineering support before making `dcc.ini` changes.

```
#############################################
# File format version
version = 1.0
#############################################
# Duration of fast advertisement after new mote joins the network
# Range:
ADV_JOIN_MOTE_TIME = 60:0
#############################################
# Duration of fast advertisement following a lost mote event.
# Range:
ADV_LOST_MOTE_TIME = 60:0
#############################################
# Min duration of fast advertisement (for manual use)
# Range:
ADV_MIN_TIME = 5:0
#############################################
# Local APD: allow APM hardware reset
# Range: 0-1
APD_ALLOW_APM_RESET = true
#############################################
# Local APD: maximum number of packet tries to APM
# Range: 2-30
APD_APM_MAX_NUM_TRIES = 3
#############################################
# Local APD: maximum queue size to APM
# Range: 1-50
APD_APM_MAX_QUEUE_SIZE = 32
#############################################
# Local APD: packet retry timeout to APM
# Range:
APD_APM_RETRY_TIMEOUT = 0:1
#############################################
# Local APD: maximum number of packet tries to dcc
# Range: 2-30
APD_CTRL_MAX_TX_TRIES = 3
#############################################
# Controller - APD: maximum number of NAK from APD
# Range: 2-30
APD_CTRL_MAX_NUM_NAK = 10
#############################################
```

```
# Local APD: apd maximum queue size (to dcc)
# Range: 1-50
APD_CTRL_QUEUE_SIZE = 5
#############################################
# Local APD: packet retry timeout (to dcc)
# Range:
APD_CTRL_PKT_RETRY_TOUT = 0:1
#############################################
# Minimum age in seconds of control session before it can be replaced by a new session
# Range: 1-31536000
API_CTL_REPLACEMENT_AGE = 600
#############################################
# Local APD: CTS poll-time
# Range:
APD_APM_CTS_POLL_TIME = 0:0.002
#############################################
# Maximum number of cli sessions allowed
# Range: 1-10
API_MAX_CLI_SESSIONS = 4
#############################################
# Maximum number of control sessions allowed
# Range: 1-100
API_MAX_CTL_SESSIONS = 100
#############################################
# Maximum number of datalog sessions allowed
# Range: 1-10
API_MAX_DATALOG_SESSIONS = 5
#############################################
# Maximum notification queue length
# Range: 20-1000
API_MAX_NOTIF_QUEUE_LENGTH = 400
#############################################
# Maximum notification sessions allowed
# Range: 1-20
API_MAX_NOTIF_SESSIONS = 5
#############################################
# Maximum time in seconds allowed to write a notification to an XML-RPC client before forcing
disconnect
# Range: 1-60
API_MAX_NOTIF_WRITE_TIMEOUT = 10
#############################################
# Maximum number of events returned in XML-RPC call to get all events
# Range: 1-5000
API_MAX_REPORTED_EVENTS = 1000
#############################################
# Maximum serial api sessions allowed
# Range: 1-10
API_MAX_SERIAL_SESSIONS = 1
#############################################
# Maximum number of API/CLI users allowed in <config><Users/></config>
# Range: 1-50
API_MAX_USERS = 20
#############################################
```

```
# Maximum webproxy sessions allowed
# Range: 1-10
API_MAX_WEBPROXY_SESSIONS = 5
#############################################
# Number of calls to calculate XML-RPC rate limit
# Range: 1-1000
API_XMLRPC_LIMIT_NUM_CALLS = 10
#############################################
# Time period to calculate XML-RPC rate limit
# Range:
API_XMLRPC_LIMIT_TIMEPERIOD = 0:1
#############################################
# Timeout (in seconds) before reporting an error when making RPC call to DCC
# Range:
API_WATCHDOG_RPC_TIMEOUT = 5
#############################################
# Maximum XML-RPC result length for a non-chunked response
# Range: 100-100000
API_XMLRPC_TRANSMIT_LENGTH = 50000
#############################################
# Time to delay flushing config file changes to flash
# Range:
CONFIG_UPDATE_DELAY = 0:5
#############################################
# Maximum time to delay flushing config file updates to flash
# Range:
CONFIG_UPDATE_TIMEOUT = 1:0
#############################################
# Max number of user(VGW) packets pending for downstream
# Range: 10-500
IOQUEUE_MAX_PEND_USER_CMDS = 100
#############################################
# Max number of controller packets in internal output queue
# Range: 1-100
IOQUEUE_MAX_CTRL_CMDS = 45
#############################################
# Max number of user packets in internal output queue
# Range: 1-100
IOQUEUE_MAX_USR_CMDS = 45
#############################################
# Refill threshold for controller packets in internal output queue
# Range: 0-100
IOQUEUE_MIN_CTRL_CMDS = 44
#############################################
# Refill threshold for users packets in internal output queue
# Range: 0-100
IOQUEUE_MIN_USR_CMDS = 30
#############################################
# To calculate ratio of controller/user packets for downstream output
# Range: 1-10
IOQUEUE_RATIO_CTRL_CMDS = 3
#############################################
# To calculate ratio of controller/user packets for downstream output
# Range: 1-10
IOQUEUE_RATIO_USER_CMDS = 1
#############################################
# Input q threshold for slowing down internal processing. 0 = disabled
# Range: 0-2000
IOQUEUE_SLOWDOWN_LOP_QSIZE = 50
#############################################
# Sleep(x) argument for slowing down at IOQUEUE_SLOWDOWN_LOP_QSIZE
# Range:
IOQUEUE_SLOWDOWN_LOP_SLEEPTM = 0:0.05
#############################################
# Number of motes to change mote discovery timeout. 0 - use fix timeout
# Range: 0-512
NET_DSCV_RATE_MOTE_THRESH = 100
#############################################
# base discovery timeout (number of oupstream frames)
# Range: 1-512
NET_DSCV_RATE_MOTE_TIME = 100
#############################################
# Controller works up to this long on decommissioning mote
# Range:
NET_DECOM_MAX_TOUT = 10:0
#############################################
# AP link feedback timeout (in Fr#1)
# Range: 1-50
NET_DOWNSTR_FEEDBACK_TOUT = 7
#############################################
# Max number downstream (FR#1) congestions
# Range: 1-100
NET_DOWNSTR_MAX_CONG = 1
#############################################
# Max number of packets to send to APM per downstream pipe link
# Range: 0-64
NET_DOWNSTR_PIPE_PK_PER_LINK = 3
#############################################
# Max number of packets to send to APM per downstream multicast link
# Range: 0-64
NET_DOWNSTR_PK_PER_LINK = 1
#############################################
# Timer sent to mote: Keep Alive Timer
# Range:
NET_MOTE_TMR_KA = 0:30
#############################################
# Timer sent to mote: Path Alarm Timeout
# Range:
NET_MOTE_TMR_PATHALARM = 0:240
#############################################
# Mote advertisement timer in steady state. 0:0 - leave same as in building state
# Range:
NET_MOTE_TMR_STEADY_ADV_RATE = 0:20
#############################################
# Enable/disable wireless security -- must be in sync with mote settings
# Range:
```

```
NET_SEC_USE_ENCRYPTION = true
##############################################
# Reject joins with bad join counter
# Range:
NET_SEC_USE_STRICT_JOIN_COUNT = true
##############################################
# Duration of time that source route alarm penalty is applied to bad path
# Range: 0-1000
NET_SRC_RT_ALARM_PENALTY_TOUT = 60:0
##############################################
# Penalty added to failed path on source route alarm
# Range: 0-65535
NET_SRC_RT_ALARM_PENALTY = 10.0
##############################################
# Source route penalty for each descendant
# Range: 0-65535
NET_SRC_RT_DESCEND_PENALTY = 0.1
##############################################
# Source route penalty for mote used in primary route
# Range: 0-65535
NET_SRC_RT_PENALTY = 10.0
##############################################
# Max UTC drift
# Range:
NET_TIME_MAX_UTC_DRIFT = 0:0.1
##############################################
# Rate at which controller sends Time Request to APM
# Range:
NET_TIME_REQ_INTERVAL = 1:0
##############################################
# Time that the master waits for AP to join before it fails over
# Range:
RDNCY_LOST_AP_TOUT = 1:0
##############################################
# Controller waits this long AP Keep Alive Timeout.
# Range:
NET_TOUT_KEEPALIVE_AP = 0:6
##############################################
# Max uninterruptable time for long operation
# Range:
NET_TOUT_MAX_CALC_TIME = 0:0.1
##############################################
# Timeout for sending repeated resets to same mote
# Range:
NET_TOUT_RESET = 1:0
##############################################
# Max time for mote state \'connected\'.
# Range:
NET_TOUT_TO_LIVE = 15:0
##############################################
# Force BW recalculation before each optimization step
# Range:
OPT_BWRECALC = false
```

```
#############################################
# Penalty for motes not connected to Fr0
# Range: 0-100
OPT_FRAME_DN_FR0_PENALTY = 10.0
#############################################
# Max penalty for path quality. Downstream frame (Fr#1)
# Range: 0-100
OPT_FRAME_DN_PATHQ_PENALTY = 2.0
#############################################
# Max penalty for path quality. Upstream frame (Fr#0)
# Range: 0-100
OPT_FRAME_UP_PATHQ_PENALTY = 4.0
#############################################
# MAX number of mote in one optimization ADD-step (0 - unlimited)
# Range: 0-100
OPT_MOTES_PER_STEP = 0
#############################################
# Coefficient to fix negative BW (FreeBW < 0 and FreeBW < -CurBW / K). 0 - only FreeBW < 0
# Range: 0-10
OPT_NEG_BW_K = 2
#############################################
# Max time for negative free BW.
# Range:
OPT_NEG_BW_TIME_THRESH = 60:0
#############################################
# Max penalty for number of links
# Range: 0-100
OPT_NUMLINKS_PENALTY = 0.4
#############################################
# Delete path after PathDown alarm received
# Range:
OPT_PATH_ALARM_DELETE = true
#############################################
# Min time between two diferent path-down alarms
# Range:
OPT_PATH_ALARM_IGNORE_TIME = 5:0
#############################################
# Timeout for deleting unused, old paths from topology
# Range:
OPT_PATH_EXPIRE_TIME = 24:0:0
#############################################
# Delete path with qualty less than given. 0 - doesn't delete path
# Range: 0-1
OPT_PATH_QUALITYTHRESHOLD_DELETE = 0.1
#############################################
# Min path quality for Pipe
# Range: 0-1
OPT_PIPE_PATHQUALITY = 0.3
#############################################
# Path quality if RSSI >= threshold
# Range: 0-1
OPT_RSSI_MAX_PATH_QUALITY = 0.75
#############################################
```

```
# Path quality if RSSI < threshold
# Range: 0-1
OPT_RSSI_MIN_PATH_QUALITY = 0.3
#############################################
# RSSI threshold, used to determine path quality
# Range: -100-0
OPT_RSSI_THRESH = -80
#############################################
# Optimization threshold between good and bad parents (0 optimization is off)
# Range: 0-1000
OPT_SCORE_THRESHOLD = 0.5
#############################################
# Timeout before changing state to Steady State
# Range:
OPT_STARTTIME = 30:0
#############################################
# Time between optimization steps
# Range:
OPT_TIMEOUT = 60:0
#############################################
# Min delay between data blocks (bcast) ms
# Range: 0-1000000
OTAP_BLK_DELAY_MIN_BC = 8000
#############################################
# Min delay between data blocks (ucast) ms
# Range: 0-10000
OTAP_BLK_DELAY_MIN_UC = 100
#############################################
# Data block size. MUST divisible by 4
# Range: 1-76
OTAP_BLK_SIZE = 76
#############################################
# AP redundant coverage multiplier
# Range: 0-1
RDNCY_AP_COV_MULTIPLIER = 0.7
#############################################
# AP redundant coverage update interval
# Range: 0-60
RDNCY_AP_COV_UPDATE_TIME = 5:0
#############################################
# Controller to Controller protocol. Max size of input queue
# Range: 10-500
RDNCY_C2C_MAX_INPUTQ_SIZE = 100
#############################################
# Controller to Controller protocol. Max size of output queue
# Range: 10-1000
RDNCY_C2C_MAX_OUTPUTQ_SIZE = 30
#############################################
# Reliable transport timeout for AP
# Range:
RELIABLE_AP_TOUT = 0:2
#############################################
# Number of retries for reliable broadcast commands
# Range: 1-60
RELIABLE_BCAST_NUM_RETR = 10
#############################################
# Range for reliable broadcast reply. N * F_UP
# Range: 1-100
RELIABLE_BCAST_REPL_RANGE = 6
#############################################
# Timeout for reliable broadcast command. N * (F_UP+F_DN)
# Range: 1-100
RELIABLE_BCAST_TOUT = 14
#############################################
# Timeout for first packet (SKActivate). x * F_UP + N * F_DN)
# Range: 1-100
RELIABLE_FRAME_DN_SK_ACK_TOUT = 15
#############################################
# Max timeout for processing SKJoin request. x * F_UP + N * F_DN)
# Range: 1-100
RELIABLE_FRAME_DN_SK_TOUT = 10
#############################################
# Timeout for first packet (SKActivate). N * F_UP + x * F_DN
# Range: 1-100
RELIABLE_FRAME_UP_SK_ACK_TOUT = 10
#############################################
# Max timeout for processing SKJoin request. N * F_UP + x * F_DN)
# Range: 1-100
RELIABLE_FRAME_UP_SK_TOUT = 10
#############################################
# Max timeout for reliable unicast command. N * (F_UP + F_DN)
# Range: 1-100
RELIABLE_MAX_TOUT = 10
#############################################
# Number of repeats for manager reliable unicast command. Use only if RELIABLE_MGR_RETR_INFINITE ==
false
# Range: 1-25
RELIABLE_MGR_UCAST_NUM_RETR = 5
#############################################
# Continuously retry commands sent on manager session. Overrides RELIABLE_MGR_UCAST_NUM_RETR
# Range:
RELIABLE_MGR_RETR_INFINITE = false
#############################################
# Min timeout for reliable unicast command N * (F_UP + F_DN)
# Range: 0-100
RELIABLE_MIN_TOUT = 0.5
#############################################
# Coefficient 'A' for round trip time calculation
# Range: 0-1
RELIABLE_RTT_CALC_A = 0.9
#############################################
# Coefficient 'B' (int) for round trip time calculation
# Range: 0-5
RELIABLE_RTT_CALC_B = 2
#############################################
# Number of repeats for VGW reliable unicast command. Use only if RELIABLE_VGW_RETR_INFINITE ==
false
# Range: 1-25
RELIABLE_VGW_UCAST_NUM_RETR = 10
#############################################
# Continuously retry commands sent on vgw session. Overrides RELIABLE_VGW_UCAST_NUM_RETR
# Range:
RELIABLE_VGW_RETR_INFINITE = true
#############################################
# Number of daily statistics intervals to store
# Range: 1-60
STATS_LONG_INTERVAL_NUM = 7
#############################################
# Number of short intervals for optimization path quality
# Range: 1-200
STATS_OPT_WINDOW = 4
#############################################
# Seconds in short statistics interval
# Range: 60-7200
STATS_SHORT_INTERVAL_LEN = 900
#############################################
# Number of short statistics intervals to store
# Range: 2-200
STATS_SHORT_INTERVAL_NUM = 96
#############################################
# Bandwidth-check process timeout
# Range:
TOP_BW_CHECK_TOUT = 1:0
#############################################
# Number of listener timeslots for advertisement frame (Fr#4) for AP. Building time
# Range: 1-20
TOP_FRAME_ADV_APLST_BLD = 6
#############################################
# Number of listener timeslots for advertisement frame (Fr#4) for AP. Steady state
# Range: 1-20
TOP_FRAME_ADV_APLST_STEADY = 2
#############################################
# Number of advertisement frame channels (Fr#4)
# Range: 1-4
TOP_FRAME_ADV_NUMCH = 1
#############################################
# Number of parents for downstream frame (Fr#1)
# Range: 1-10
TOP_FRAME_DN_PARENTS = 2
#############################################
# Range for selecting next link (Fr#1)
# Range: 1-100
TOP_FRAME_DN_TS_RANGE = 40
#############################################
# Max number of hops in pipe
# Range: 1-7
TOP_FRAME_PIPE_NUMHOPS = 7
#############################################
# Check power information prior to building pipe
# Range:
TOP_FRAME_PIPE_SAFETYCHECK = true
#############################################
# Minimum size of pipe frame (Fr#2,3)
# Range: 1-4096
TOP_FRAME_PIPE_SIZE = 16
#############################################
# Number of links for upstream frame(Fr#0)
# Range: 1-10
TOP_FRAME_UP_LINKS = 4
#############################################
# Number of parents for upstream frame (Fr#0)
# Range: 1-10
TOP_FRAME_UP_PARENTS = 2
#############################################
# Max number of parents for upstream frame (Fr#0). 0 - unlimited
# Range: 0-100
TOP_FRAME_UP_PARENTS_MAX = 8
#############################################
# Oversubscribe coefficient for link (1.0 no oversubscribing
# Range: 1-100
TOP_LINK_OVRSUBSCR = 3.0
#############################################
# Max AP bandwidth
# Range: 1-100
TOP_MAX_AP_BW = 72.0
#############################################
# Default value for max number of mote's links. Overwritten when reported by mote
# Range: 10-1000
TOP_MAX_LINKS = 10
#############################################
# MAX number of motes in network
# Range: 10-NUMBACTMOTE_MAX
TOP_MAX_MOTES = 501
#############################################
# Default value for maximum number of mote's neighbors
# Range: 4-1000
TOP_MAX_NEIGHBORS = 32
#############################################
# Number of AP multicast links (Fr#1)
# Range: 0-64
TOP_NUM_DN_MCAST_LINKS = 8
#############################################
# Time to resolve service request
# Range:
TOP_SERVICE_REQ_TOUT = 2:0
#############################################
# Min path quality for good neigbor
# Range: 0-1
TOP_GOODNBR_QUALITY = 0.6
#############################################
# Max number upstream link for add jitter to select timeslot
# Range: 0-100
```

```
LINKJITTER_NUMLINKTH = 4
```

4.11.6 Default platform.ini

This file maps access point GPIO pins to Linux host system GPIO, tells where logs are kept, and what serial port is used to communicate with the Access Point.

```
#############################################
# File format version

version = 1.0
#############################################
# Reset pin
# String Length Range: 0-128
GPIO_RESET = /dev/gpioPB17
#############################################
# Time pin
# String Length Range: 0-128
GPIO_TIME = /dev/gpioPB21
#############################################
# Serial port clear-to-send pin
# String Length Range: 0-128
GPIO_SP_CTS = /dev/gpioPC12
#############################################
# Mote clear-to-send pin
# String Length Range: 0-128
GPIO_MT_CTS = /dev/gpioPB19
#############################################
# Serial port ready-to-send pin
# String Length Range: 0-128
GPIO_SP_RTS = /dev/gpioPB16
#############################################
# Mote ready-to-send pin
# String Length Range: 0-128
GPIO_MT_RTS = /dev/gpioPB18
#############################################
# Access point GPIO, unused
# String Length Range: 0-128
GPIO_AP_GPIO = /dev/gpioPB30
#############################################
# Access point flash disconnect (unused)
# String Length Range: 0-128
GPIO_AP_FL_DIS = /dev/gpioPB31
#############################################
# Access point power
# String Length Range: 0-128
GPIO_AP_POWER = /dev/gpioPC6
#############################################
# Join LED
# String Length Range: 0-128
GPIO_JOIN = /dev/gpioPC1
#############################################
# Subscription LED
# String Length Range: 0-128
GPIO_SUBSCRIPTION = /dev/gpioPC0
#############################################
# Serial port transmit
# String Length Range: 0-128
GPIO_SP_TX = /dev/gpioPB12
#############################################
# Serial port receive
# String Length Range: 0-128
GPIO_SP_RX = /dev/gpioPB13
#############################################
# Location of dust log files (make sure to end with '/')
# String Length Range: 0-128
PATH_LOG_LOCATION = /opt/dust-manager/log/
#############################################
# Local APD serial port
# String Length Range: 0-128
TTY_LOC_APD = /dev/ttyS2
```

4.11.7 Default system.ini

This file configures log sizes and a number of ports and addresses for operation, diagnostic functions, and redundancy.

For those migrating from earlier (<4.0) versions of the SmartMesh WirelessHART Manager, note that some default values may have changed.

Changing ports here (e.g. moving the XML-RPC port, PORT_CTRL_TCP, from the default 4445) may render your manager inoperable if you don't also configure the firewall (via firewall.conf) to allow the same ports.

```
#############################################
# File format version
version = 1.0
#############################################
# If true, manager runs in embedded mode (security keys are exposed in APIs
# Range: true, false
# MODE_EMBEDDED = FALSE
#############################################
# Maximum size of dcc.log file (in bytes).
# Range: 1000-1000000
# LOGS_DCC_LOG_SIZE = 400000
```

```
#############################################
# Maximum size of event.log file (in bytes).
# Range: 1200-1200000
# LOGS_EVENT_LOG_SIZE = 480000
#############################################
# IP address of remote-log server
# String Length Range: 0-100
# LOGS_RLOG_IPADDR =
#############################################
# Port number of remote-log server
# Range: 1-65535
# PORT_RLOG = 4545
#############################################
# Port for APD interface
# Range: 1024-65535
# PORT_CONTROL_APD = 60000
#############################################
# Input port for redundancy link
# Range: 1024-49000
# PORT_CONTROL_C2C_IN = 5000
#############################################
# Output port for redundancy link
# Range: 1024-49000
# PORT_CONTROL_C2C_OUT = 5000
#############################################
# Accept local connections only (c2c, xml-rpc, apd)
# Range: true, false
# PORT_CONTROL_USE_LOCAL = false
#############################################
# Port number for xml-rpc control channel
# Range: 1024-65535
# PORT_CTRL_TCP = 4445
#############################################
# Port number for xml-rpc notification channel
# Range: 1024-65535
# PORT_NOTIF_TCP = 24112
#############################################
# Use SSL for xml-rpc connection
# Range: true, false
# USE_SSL_PORTS = false
#############################################
# SSL. Port number for xml-rpc control channel
# Range: 1024-65535
# PORT_CTRL_SSL = 4446
#############################################
# SSL. Port number for xml-rpc notification channel
# Range: 1024-65535
# PORT_NOTIF_SSL = 24113
#############################################
# APD CLI interface port
# Range: 1024-65535
# PORT_LOC_APD_CLI = 55551
#############################################
```

```
# APD local bind port for connection with DCC
# Range: 1024-65535
# PORT_LOC_APD_CONTROL = 60002
#############################################
# Port number for logging daemon
# Range: 1024-65535
# PORT_LOGGER = 21333
#############################################
# Port number for nwconsole server
# Range: 1024-65535
# PORT_NWCONSOLE = 49004
#############################################
# Port number for watchdog server
# Range: 1024-65535
# PORT_WATCHDOG = 49007
#############################################
# Redundancy link: rate of KA messages (when there's no traffic)
# Range: N/A
# RDNCY_C2C_KA_TIMEOUT = 0:1
#############################################
# Redundancy link: max number of nacks before failure is declared
# Range: 1-30
# RDNCY_C2C_MAX_NUM_NACKS = 10
#############################################
# Redundancy link: packet tx failure timeout
# Range: N/A
# RDNCY_C2C_TX_FAILURE_TIMEOUT = 0:0.6
#############################################
# Redundancy link: packet retry timeout upon receiving nack
# Range: N/A
# RDNCY_C2C_NACK_RETRY_TIMEOUT = 0:0.1
#############################################
# If true, start controller in standalone mode.
# Range: true, false
# RDNCY_STANDALONE_MODE = true
#############################################
# Peer controller's IP Address
# String Length Range: 7-15
# RDNCY_PEER_IP_ADDRESS = 127.0.0.1
#############################################
# Time to wait for promotion after losing connection with Master. 0:0 - promote immediately after
disconnect
# Range: N/A
# RDNCY_PROMO_TIME = 1:0
#############################################
# Path for API notification socket
# String Length Range: 1-128
# API_NOTIFICATION_PATH = notif.sock
```

4.12 Datalog Utility

4.12.1 Overview

The datalog utility is a component of the SmartMesh WirelessHART Manager that provides a simple mechanism for capturing, storing, and reporting the contents of data packets received from the mote network. Datalog commands can be run as the `dust` user from the Manager's Linux command line.

> Testing of a variety of SD cards from several manufacturers has shown that some SD cards will fail to properly respond during the card identification mode, defined in "SD Specifications, Part 1, Physical Layer Simplified Specification, Version 4.10, January 22, 2013" following power on of the LTP5903. As a result the SD Card is not recognized, and an LTP5903 system recovering from a loss of power cannot guarantee that the datalog utility will be able to start up automatically. The issue appears to result in the SD Card entering an invalid state that it cannot recover from without the card being removed from the LTP5903's SD Card socket. In addition, cards that fail to negotiate the card identification sequence do not fail consistently, so determining exactly which cards do or do not suffer from this issue is not tractable. During testing, several cards never failed, but as we are not able to identify the root cause and then be able to define a definitive pass / fail test case, Linear Technology cannot guarantee that any card will operate correctly through a power on sequence. Testing has never discovered a card that would not function correctly when inserted after the LTP5903 has completed its boot sequence.

4.12.2 Using Datalog

Starting a Capture

Before starting a capture, make sure an SD card is inserted into the Manager's SD/MMC port. The Manager automatically mounts the filesystem on an SD card when the card is inserted.

To start a capture, from the Manager's Linux command line, type:

```
datalog start <capture-id>
```

The capture id is an arbitrary identifier for the capture. The identifier is used in the name of the capture file and in the status and stop commands to refer to a capture operation. There are several optional parameters:

- `-t` – indicates that packet timestamps should be included in the capture file.
- `-h <header-file>` – indicates that the given file is associated with the capture and should not be deleted. The header file should be stored in the capture directory.

For example, the following command will start a capture with the id *mycapture* that includes timestamps and has an associated file named *myheader.txt* in the capture directory.

```
datalog start mycapture -t -h myheader.txt
```

For historical reasons, all files in the capture directory that are not associated with the new capture will be deleted when a capture starts. To reiterate, if you forget to include the `-h <header>` option, any existing header file will be deleted.

Stopping a Capture

When a capture is stopped, a fixed-length trailer is appended to the file. For details, see the Datalog Output section.

To stop a capture, type:

```
datalog stop <capture-id>
```

After the capture is stopped, the SD card filesystem can be unmounted so that the SD card can be removed to transfer the data to another computer.

The `umount` command cleanly unmounts the filesystem of the SD card to ensure the OS has written all pending data to the SD card.

```
dust@manager:/media/card$ cd
dust@manager:~$ sudo umount /media/card
```

Normally, the `umount` command produces no output.

> If an error message appears indicating that the card can not be unmounted because it is busy, then there may be a capture process still running. The same error appears if a login shell has navigated to the SD card (`/media/card`).umount: /media/card: device is busy.
> (In some cases useful info about processes that use
> the device is found by lsof(8) or fuser(1)

Capture Status

To see the status of a capture, type:

```
datalog status <capture-id>
```

The `capture-id` parameter is optional. If no capture id is specified, the `status` command will show the status of any capture that is in-progress. The `status` command shows:

- the capture id,
- the full path of the capture file,
- the name of the header file (if specified),
- the time the capture was started, and
- the size of the capture and the remaining storage available in the capture directory.

```
$ datalog status 101
Capture '101'
   Capture file: /media/card/capture.101
   Started:      Thu Oct 25 11:55:30 2012
   Size:         0 kB (remaining: 15538848 kB)
```

File Transfer

Capture files can be downloaded from the Manager at any time using one of the file transfer protocols supported by SSH: SCP or SFTP. These mechanisms can also be used to upload header files to the capture directory. If the capture file is downloaded while the capture is in progress, the end of the file may contain an incomplete data record. At any given time, there will be at most one incomplete record when the capture file is retrieved. The incomplete record will be completed when data is next written to the capture file.

Configuration

Datalog has several configuration parameters that are stored in a configuration file in `/opt/dust-manager/conf/settings/datalog.conf`:

- `CAPTURE_DIR` – The directory for storing capture files and header files. This directory must be writable by the 'dust' user. If the directory does not exist, datalog will try to create it.
  Defaults to `/media/card`
- `CAPTURE_PREFIX` – A prefix for the name of capture files. The name of the capture file will be `<prefix>.<capture-id>`.
  Defaults to *capture*
- `CAPTURE_DEVICE` – The device to use for captures. This is used for checking remaining memory available for captures.
  Defaults to `/dev/mmcblk0p1`
- `PID_FILE` – The filename for storing the process id file. For internal use only.
  Defaults to `/opt/dust-manager/var/dust-datalog.pid`

4.12.3 Datalog Output

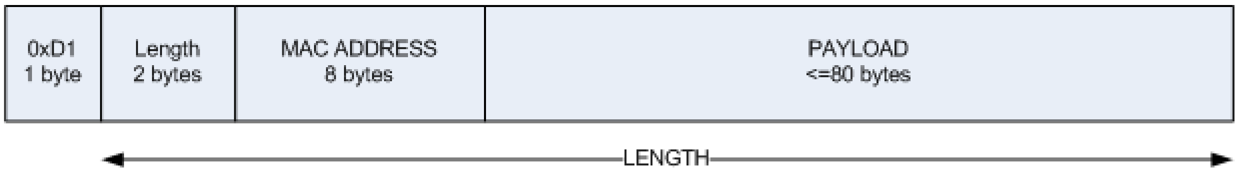
As the data logging operation runs, it captures the data received from motes and continuously appends to a capture file. The capture file is named `capture.ID`, where ID is the capture identifier from the start operation. The capture file consists of a sequence of data records, and is terminated with a trailer when the data logging operation is stopped. Each data record is variable sized, binary data captured from a single serial packet transmitted from a mote. The time field is optionally present depending on whether the timestamp parameter was passed to the start command. The optional header file is not part of the capture file; it is simply a file stored in the same capture directory. The contents of this header file are not parsed by the datalog tool and may contain arbitrary data.

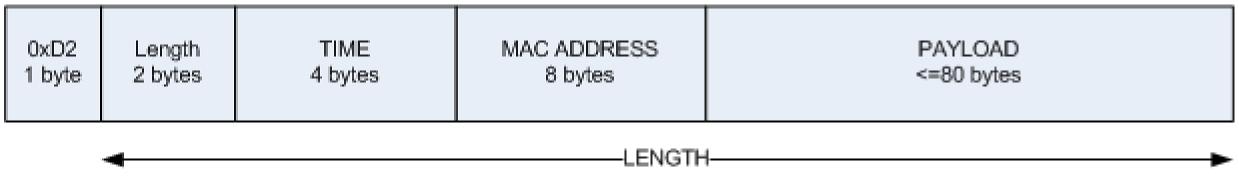
The time field present in data records and the trailer is in Unix time format: the number of seconds since midnight, Jan 1, 1970. All multi-byte numbers generated by the datalog tool (i.e., the length and time values) are stored in the capture file in "little endian" format, with the least significant byte first. Any data in the payload of the serial data packet is stored as it is received. The format of the data payload is opaque to the `datalog` tool.

Format of Data Records

A magic number at the front of the data packet is used as a tag to describe the contents of the record. The length field contains the length in bytes of the remainder of the record, including the length field.

- `0xD1` indicates that no timestamp is present.
- `0xD2` indicates that the timestamp is present.



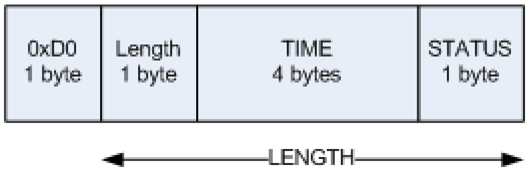


Format of Trailer

When a capture is stopped (either by the `stop` command or by automatically closing due to size constraints), a fixed-length trailer with the following information is appended to the capture file:

- Magic number (`0xD0`)
- Timestamp at which the capture was stopped
- Capture status

The trailer allows the client to determine whether the capture was completed properly. A magic number is present so that a correctly formed trailer can be detected.



The capture status indicates the reason for finalizing the capture file. The status is one of:

- **Status** = 0: OK. The capture was terminated by a client stop command.
- **Status** = 1: Size Error. The capture was stopped because of size constraints in the capture directory.

Note: the **Length** field is 1 byte, which differs from 2-byte **Length** field in Data Records. This is done for historical backward compatibility reasons.

Converting captured data to ASCII

The `datalogConvert` tool is used to convert capture files into a text format. In the text format, MAC addresses and data payloads are represented in hexadecimal; timestamps (if present) are represented as *MM/DD/YY HH:MM:SS*. The resulting file can be imported by programs that accept comma separated values, such as MS Excel. The converted files are much larger than the standard capture files, so make sure there is plenty of space on the manager for the output before using this tool.

Usage

From the Manager's Linux command line:

```
datalogConvert /media/card/capture.101 /media/card/capture.101.csv
```

This command will take the file in `capture.101` and convert it to a comma-delimited text format which will be stored in `capture.101.csv`.

4.12.4 How to Restart Datalog on System Reboot

Overview

This document describes how to modify the datalog script to restart automatically if the Manager reboots.

There are several components of this project to understand:

- Detect an in-progress capture (and its id)
- Prevent clearing the capture directory
- Run a script on startup

The code presented in this document is presented for example only. It has not been tested.

Detect an in-progress capture

The simplest way to detect an in-progress capture is to look for a well-known file that is written (by the `datalog` script) when the capture starts. The `datalog` script must write to this file when a capture starts (at the end of the start action). The `datalog` script must check for this file in the resume action.

Another aspect of detecting an in-progress capture is to ensure that the resumed capture is associated with the same capture id.

Another aspect of detecting an in-progress capture is to ensure that the new capture file does not overwrite any previous captures. The name of the new capture file should not be exactly the same as the previous capture.

```
# Add a variable for the file name that indicates a capture is running
CAPTURE_RUNNING=.capture_id
```

```
# In the resume action, check whether the "capture in-progress" file exists
if [ -f $CAPTURE_DIR/$CAPTURE_RUNNING ]; then

  # The presence of this file indicates that we should resume the capture

  # Read the past capture id
  LAST_CAPTURE_ID=`cat $CAPTURE_DIR/$CAPTURE_RUNNING`

  # [Add Code] determine the new capture filename

  # [Add Code] perform similar operations to the start actions to initiate a capture

fi
```

```
# At the end of the start code, write the capture id to the CAPTURE_RUNNING file
echo "$CAPTURE_ID" > $CAPTURE_DIR/$CAPTURE_RUNNING
```

Prevent deleting any previous captures

By default, the `datalog` script clears the capture directory on startup. If we are resuming a capture, we do not want to delete the previous capture logs. The easiest solution is to just remove the lines that clear the capture directory.

```
case "$action" in
    start)
        ...

        # clear out the capture directory before starting
        # [REMOVE] rm -f ${CAPTURE_DIR}/* 2>>/dev/null
        # [REMOVE] rm -f ${CAPTURE_DIR}/.* 2>>/dev/null

        ...
```

Given that the typical SD card capacity has increased, the user may consider not clearing the capture directory on each capture start.

Startup scripts

System startup scripts are stored in `/etc/init.d/` and symlinked to the appropriate runlevels. This setup is accomplished with the `update-rc.d` command line utility.

The startup script should have a command line argument that indicates whether the system is starting up or shutting down. Here is a simple example of a startup script that calls `datalog resume` as the `dust` user when the system boots.

```
#!/bin/sh
#
# Run the datalog resume command on startup

DATALOG=/opt/dust-manager/bin/datalog

case "$1" in
    start)
        echo -n "Running datalog resume script... "
        su dust - $DATALOG resume
        echo "done."
        ;;

    stop)
        # nothing to do here
        :
        ;;

    *)
        echo "usage: /etc/init.d/datalog-resume {start|stop}"
        exit 1
        ;;
esac

exit 0
```

For the purposes of this project, it's better not to put too much intelligence into the startup script. Instead, the common configuration (paths, file naming conventions, etc) can live in the `datalog` script itself. So the main job of the startup script is to call the `datalog` script with an argument that tells it to try to resume a previous capture.

The startup script for the persistent captures should run after the Manager software is started. We ensure this by running the startup script last in the startup sequence (99).

```
root@manager# update-rc.d -f resume-datalog defaults 99 1
```

4.13 Configuring PPP on Serial 1

By default, the Manager is configured to allow Linux command line (CLI) login on the serial port labelled **Serial 1**. It is also possible to configure **Serial 1** to support PPP access. In this mode, **Serial 1** is always expecting a connection from a PPP client.

4.13.1 Switching Serial Port Modes

The following commands will switch the mode of the **Serial 1** port:

```
dust@manager$ sudo config-login disable
Disabling login on ttyS1
INIT: Sending processes the TERM signal

dust@manager$ sudo config-ppp enable
Enabling PPP on /dev/ttyS1.
Loading ppp modules.
PPP generic driver version 2.4.2
PPP BSD Compression module registered
PPP Deflate Compression module registered
update-rc.d: /etc/init.d/ppp-modules exists during rc.d purge (continuing)
 Removing any system startup links for ppp-modules ...
 Adding system startup for /etc/init.d/ppp-modules.
```

4.13.2 PPP Configuration

The default PPP configuration expects a client to connect at 115200 baud, 8N1, with no flow control. When connected, the default PPP configuration configures the Manager on IP address `192.168.101.10` and the PPP client on IP address `192.168.101.11`.

4.13.3 Linux Client Configuration

Linux distributions provide a PPP package that can interoperate with PPP running on the Manager.

1. Install the `ppp` package on the Linux client.
2. Connect the Linux client to the Manager's Serial 1 port with a serial cable.
3. Add the following configuration to `/etc/ppp/options.<port>` where `<port>` is the serial device name on the Linux client, e.g. `ttyS1`.

```
noccp
deflate 15,15
bsdcomp 15,15
nocrtscts
115200
192.168.101.10:192.168.101.11
```

4. Run `pppd` to connect to the Manager.

```
$ sudo pppd /dev/<port>
```

5. To disconnect, kill the `pppd` process.

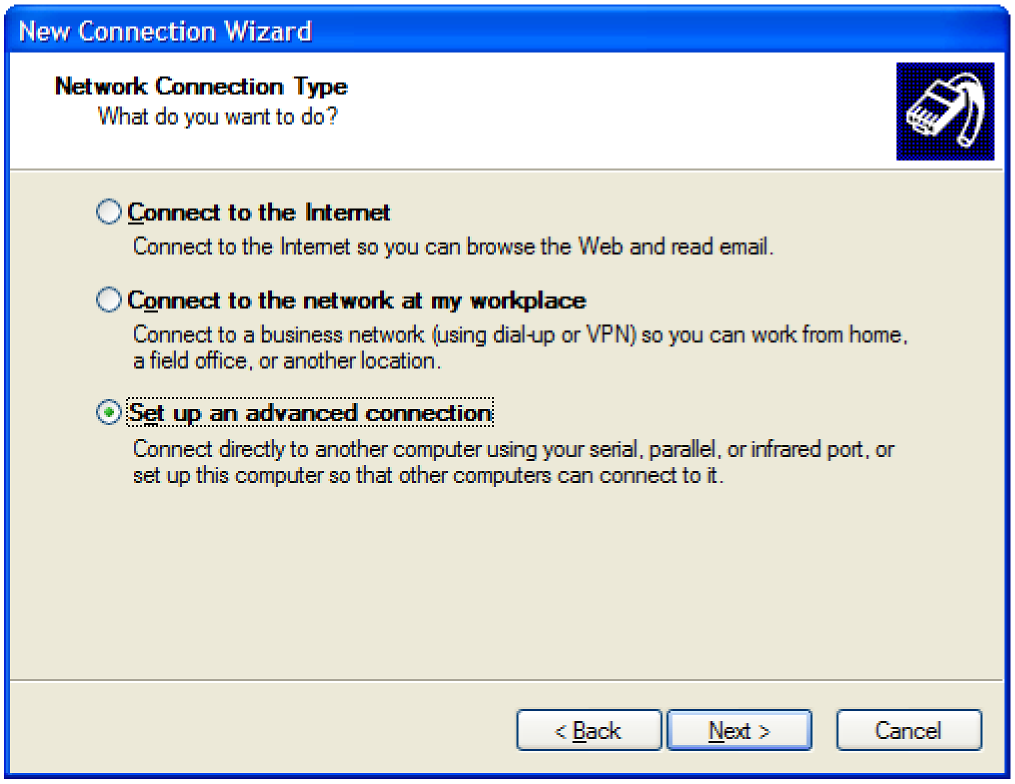
4.13.4 Windows Client Configuration

The built-in Windows PPP connection expects an extra handshake from the PPP server. To inter-operate with Windows, the PPP connection must be configured to expect a Windows client, use:

```
dust@manager$ sudo config-ppp enable-windows
```

On the Windows machine, open **Network Connections** under the **Control Panel** and create a **New Network Connection**. The following steps walk through the New Network Connection wizard.

Select **Set up an advanced connection**:



Network Connection Type

What do you want to do?

( ) **Connect to the Internet**

Connect to the Internet so you can browse the Web and read email.

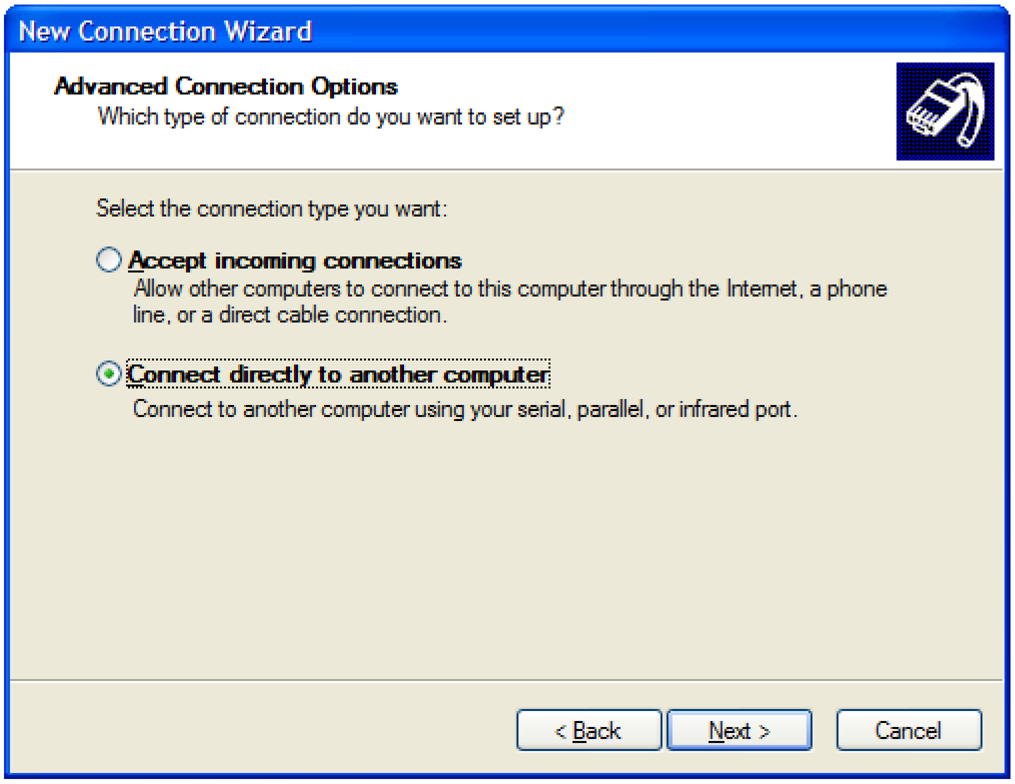
( ) **Connect to the network at my workplace**

Connect to a business network (using dial-up or VPN) so you can work from home, a field office, or another location.

[✓ **Set up an advanced connection**]

Connect directly to another computer using your serial, parallel, or infrared port, or set up this computer so that other computers can connect to it.

Select **Connect directly to another computer**:


New Connection Wizard

Advanced Connection Options

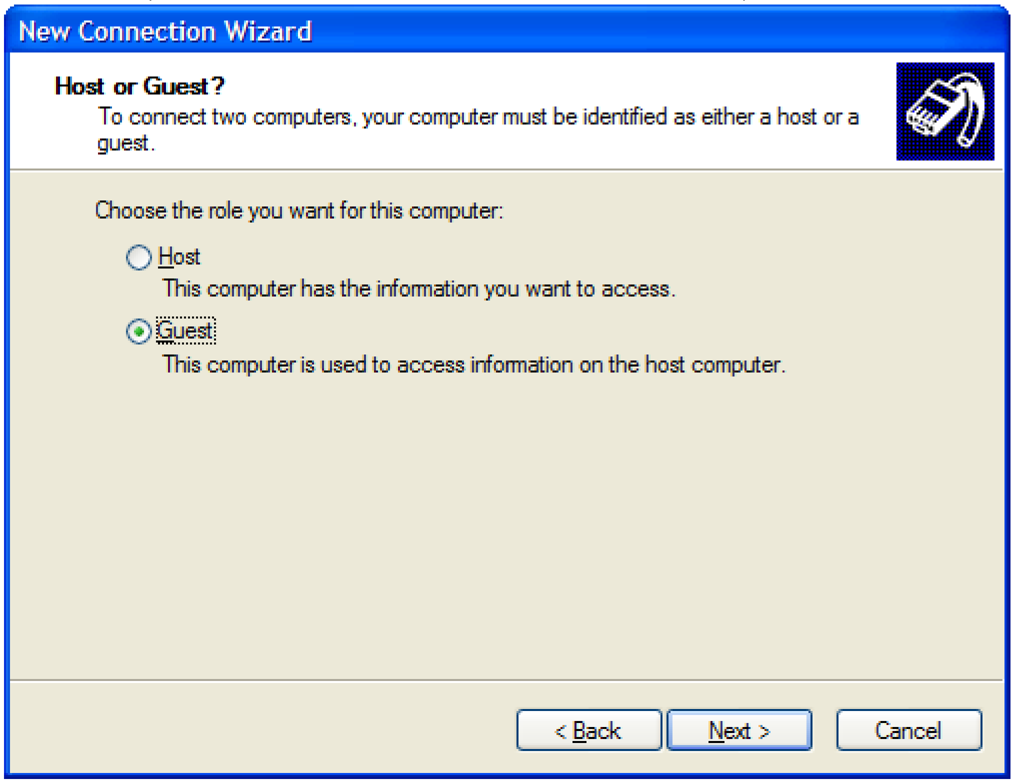
Which type of connection do you want to set up?

Select the connection type you want:

( ) **Accept incoming connections**
Allow other computers to connect to this computer through the Internet, a phone line, or a direct cable connection.

[✓ **Connect directly to another computer**]
Connect to another computer using your serial, parallel, or infrared port.

Select **Guest** (because the Windows machine will act as the PPP client):



New Connection Wizard

Host or Guest?

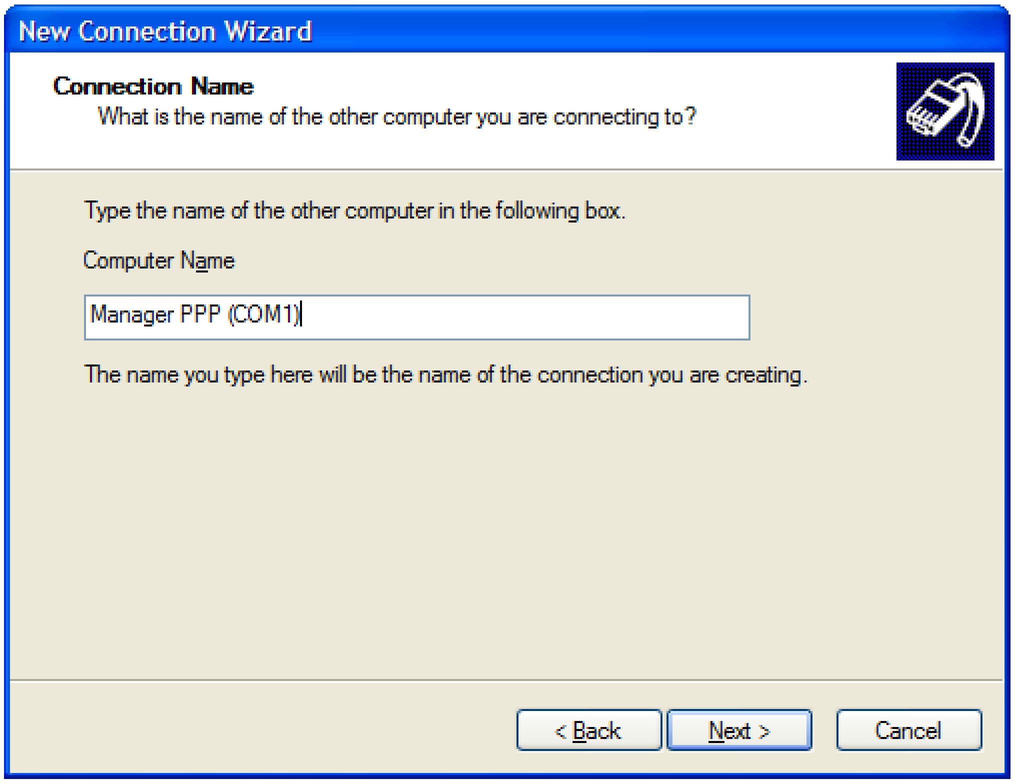
To connect two computers, your computer must be identified as either a host or a guest.

Choose the role you want for this computer:

( ) Host
This computer has the information you want to access.

[✓ Guest]
This computer is used to access information on the host computer.

Enter a meaningful name for this connection. It is useful to add the serial port that this connection will be used on.


New Connection Wizard

Connection Name

What is the name of the other computer you are connecting to?

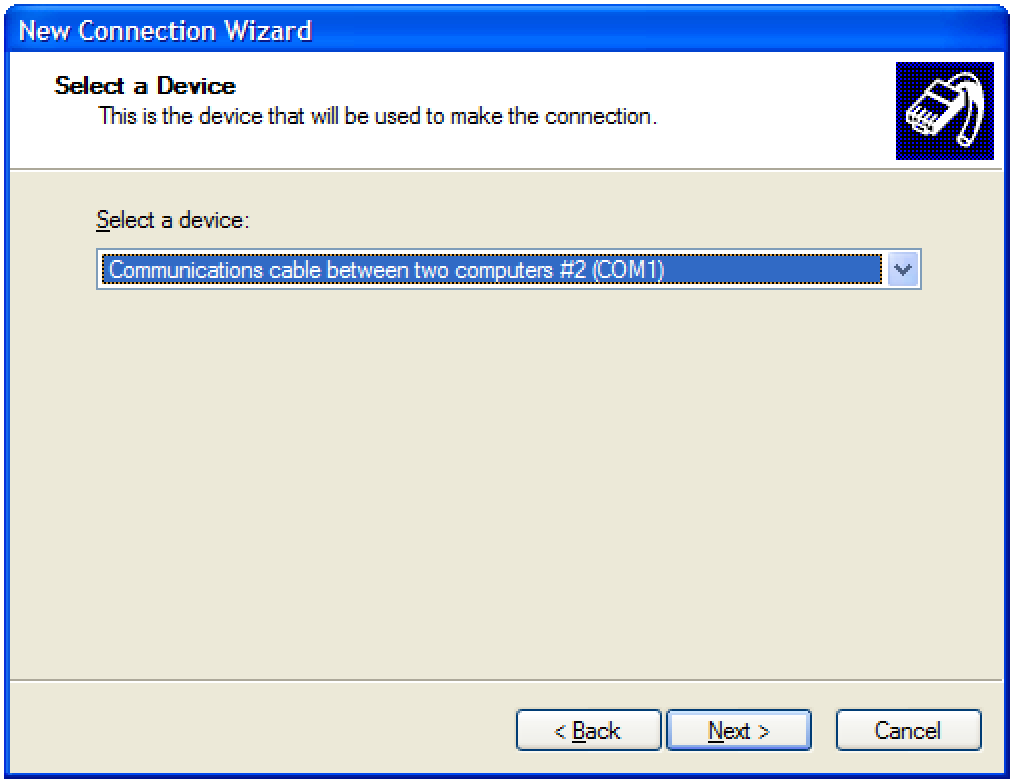
Type the name of the other computer in the following box.

Computer Name

Manager PPP (COM1)

The name you type here will be the name of the connection you are creating.

Select the serial port over which the computer is connected to the Manager **Serial 1** port.



New Connection Wizard

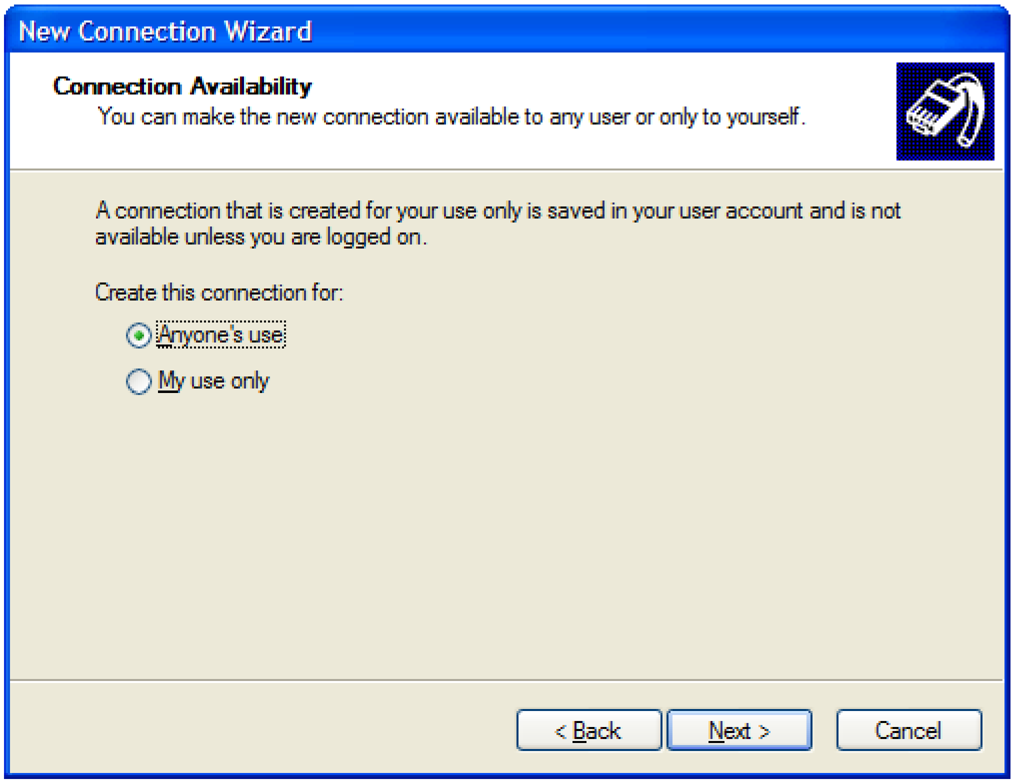
Select a Device

This is the device that will be used to make the connection.

Select a device:

Communications cable between two computers #2 (COM1)

Select **Anyone's use** to allow any user of this computer to use this PPP connection.



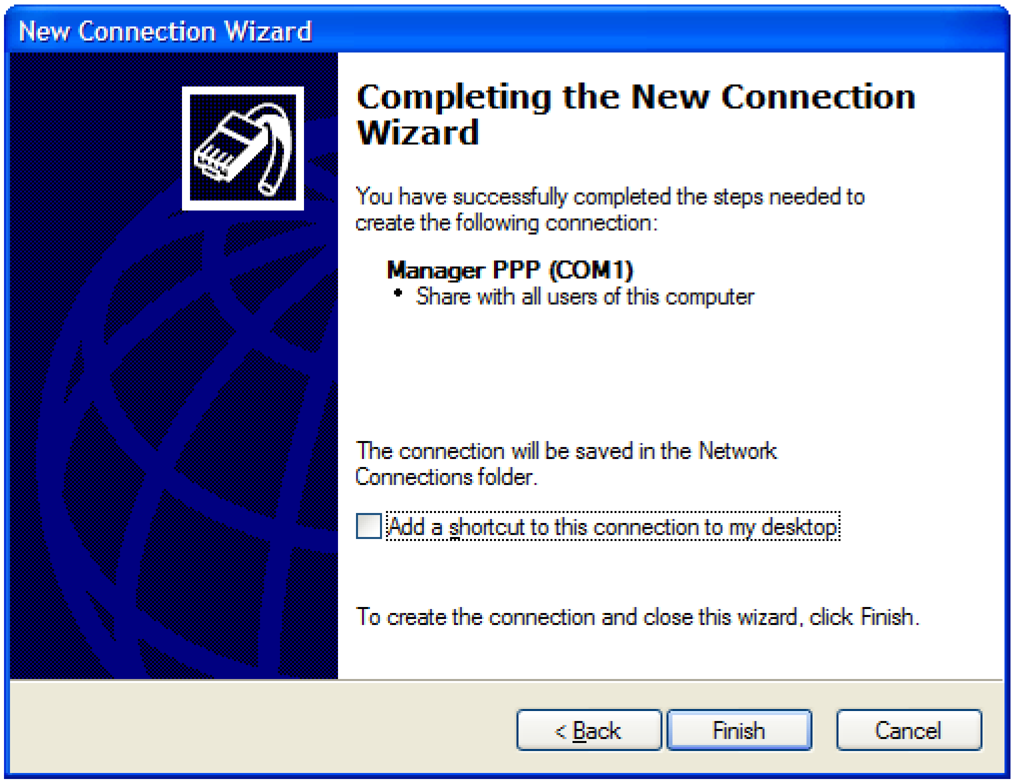
Connection Availability
You can make the new connection available to any user or only to yourself.

A connection that is created for your use only is saved in your user account and is not available unless you are logged on.

Create this connection for:

[✓ Anyone's use]
My use only

Select **Finish** to complete the wizard.



New Connection Wizard

Completing the New Connection Wizard

You have successfully completed the steps needed to create the following connection:

Manager PPP (COM1)

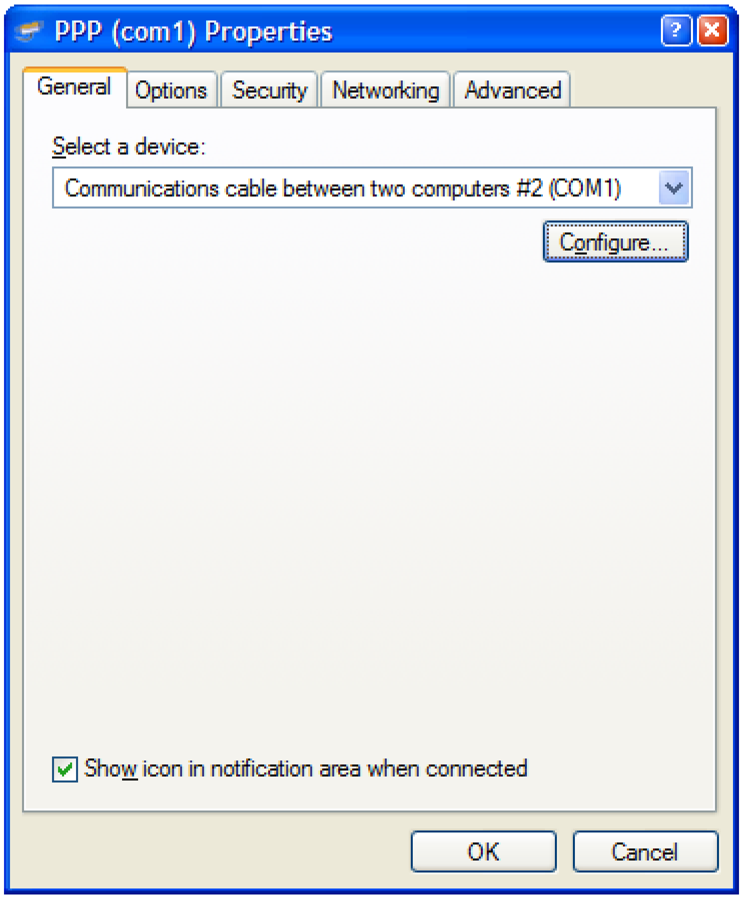
- Share with all users of this computer

The connection will be saved in the Network Connections folder.

☐ Add a shortcut to this connection to my desktop

To create the connection and close this wizard, click Finish.

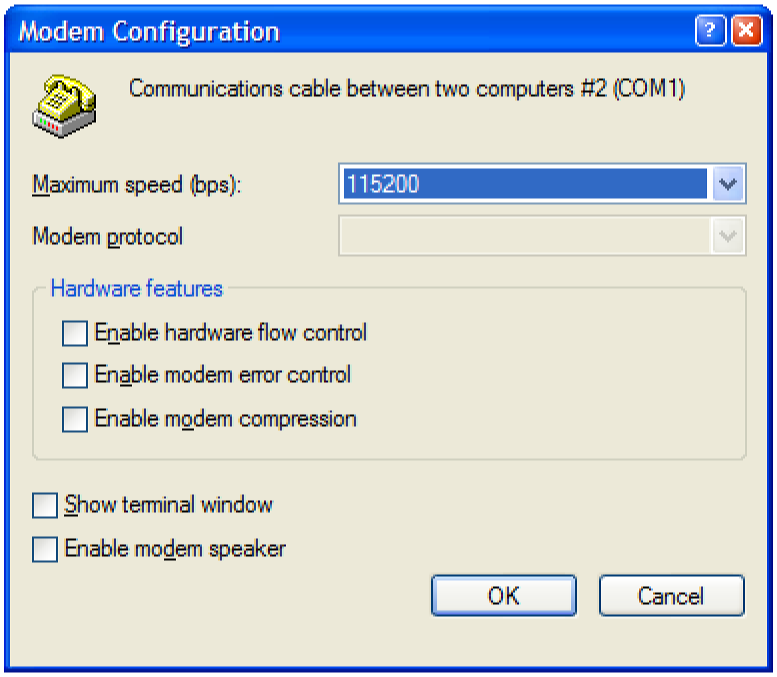
There are a couple of additional steps to configure the PPP connection properly.Right click on the PPP connection icon and open the **Properties** dialog.



Select a device:
Communications cable between two computers #2 (COM1)

✓ Show icon in notification area when connected

Under the **General** tab, select **Configure...** to configure the serial port properties. Make sure that the speed is set up 115200 baud and hardware flow control is disabled.


Modem Configuration

Communications cable between two computers #2 (COM1)

Maximum speed (bps): 115200

Modem protocol

Hardware features

☐ Enable hardware flow control

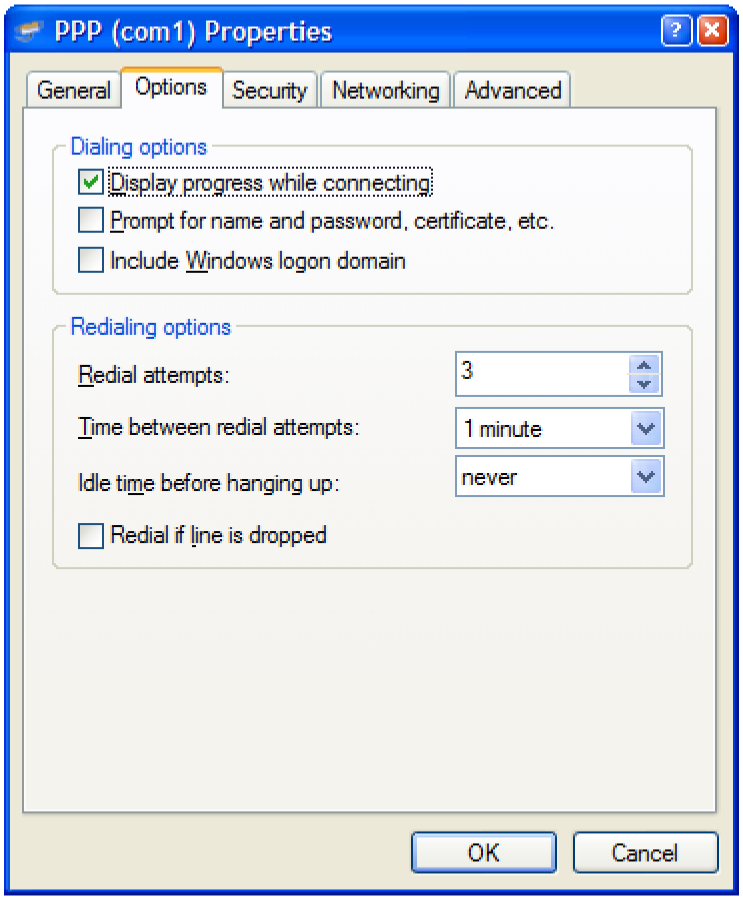
☐ Enable modem error control

☐ Enable modem compression

☐ Show terminal window

☐ Enable modem speaker

Under the **Options** tab, make sure **Prompt for name and password, etc** is unchecked.


PPP (com1) Properties

Dialing options

[✓] Display progress while connecting

[ ] Prompt for name and password, certificate, etc.

[ ] Include Windows logon domain

Redialing options

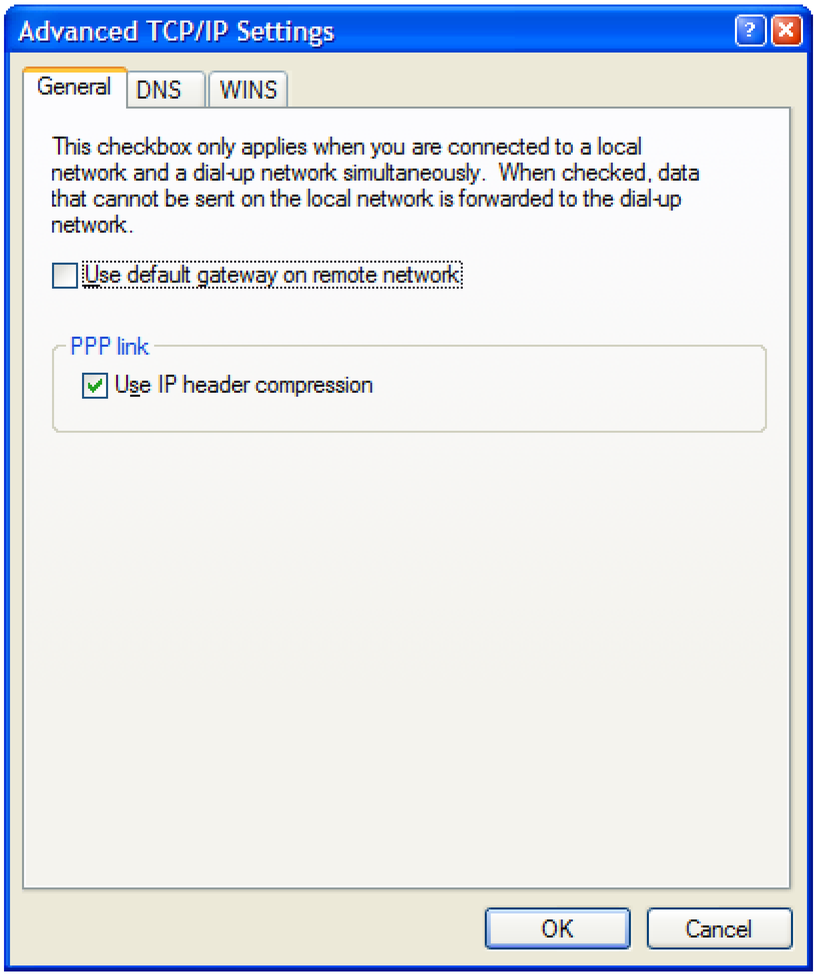
Redial attempts: 3

Time between redial attempts: 1 minute

Idle time before hanging up: never

[ ] Redial if line is dropped

Under the **Networking** tab, select **Internet Protocol (TCP/IP)** and then select **Properties**. In the **Internet Protocol (TCP/IP) Properties** dialog, select **Advanced…**. In the **Advanced TCP/IP Settings** dialog deselect **Use default gateway on remote network**.



This checkbox only applies when you are connected to a local network and a dial-up network simultaneously. When checked, data that cannot be sent on the local network is forwarded to the dial-up network.

☐ Use default gateway on remote network

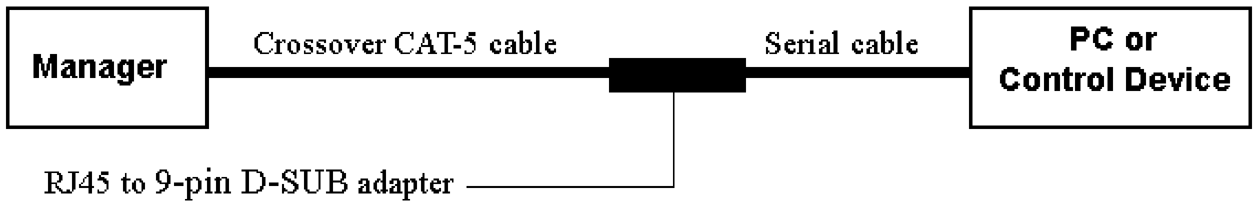
PPP link

✓ Use IP header compression

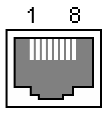
Once the Windows PPP connection is set up, double click on the connection icon to start the connection.

4.13.5 Assembling a 9-pin D-SUB Adapter for Serial 1

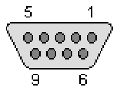
If you are connecting the **Serial 1** interface on a LTP5903CEN-WHRmanager to a 9-pin D-SUB RS232 port on a PC or control device, you need to use an RJ45 to 9-pin D-SUB RS232 adapter.



A crossover Ethernet cable is used to connect the **Serial 1** interface to the adapter. The adapter in the following example is a female RJ45 to female 9-pin D-SUB. The arrows indicate how to match up the RJ45 and 9-pin D-SUB connector pins.



Female RJ45 Connector



9-pin D-SUB Connector

RJ45 Pinout		9-pin D-SUB Pinout	
Pin	Signal Description	Pin	Signal Description
1	TX out of manager	3	TXD
2	RTS out of manager	7	RTS
3	RX into manager	2	RXC
4	GND	4	Not connected
5	GND	5	GND
6	CTS into manager	8	CTS
7	Not connected	7	RTS
8	GND	8	CTS
		9	Not connected

Matching Up RJ45 and 9-pin D-SUB Connector Pins

4.14 Manager Redundancy

4.14.1 Overview

The SmartMesh WirelessHART Manager (>=4.1.0) includes Linux-HA hooks to support hot-failover redundancy via the mgrctl script. The Linux-HA components themselves are not supplied with the manager.

When redundancy is enabled in the Manager:

1. When the Manager starts, it starts in slave mode and waits to be promoted to master. In a redundant system, promotion is handled by the redundancy manager, an external process that manages monitoring and failover.
2. The Manager watchdog process will initiate a ''failover'' rather than restart if a critical failure occurs. The watchdog notifies the external redundancy manager to initiate failover.

3. The master Manager listens for a connection from the slave Manager to exchange configuration and state. The slave Manager tries to connect to its peer (master). Once connected, the slave keeps the connection open and receives periodic updates from the Manager.

The redundant peers must be able to connect to each other over TCP.

4.14.2 Configuration

Certain configuration parameters must be changed from their default values to enable redundancy.

On each Manager, edit */opt/dust-manager/conf/config/system.ini:*

```
##############################################
# If true, start controller in standalone mode.
# Range: true, false
RDNCY_STANDALONE_MODE = false
##############################################
# Peer controller's IP Address
# String Length Range: 7-15
RDNCY_PEER_IP_ADDRESS = <IP address of peer>
##############################################
```

On both the master and slave, the RDNCY_PEER_IP_ADDRESS must point to the IP address of the other Manager.

After these configuration parameters are changed, the Manager must be restarted for the new parameters to take effect.

With RDNCY_STANDALONE_MODE set to false, the Manager starts in slave mode and waits for the redundancy manager (or user) to promote the Manager to be the master.

```
sudo /etc/init.d/dust-manager start
sudo /etc/init.d/dust-manager switch
```

4.14.3 Advanced configuration

Additional configuration parameters can be adjusted for system-specific configuration.

/opt/dust-manager/conf/config/system.ini

```
##############################################
# Input port for redundancy link
# Range: 1024-49000
# PORT_CONTROL_C2C_IN = 5000
```

It's possible to change the port on which the master listens for connections from a slave.

4.14.4 Firewall configuration

If the TCP port used for redundancy is changed, the firewall configuration in */etc/init.d/firewall* must be changed as well.

5 The SmartMesh WirelessHART Mote

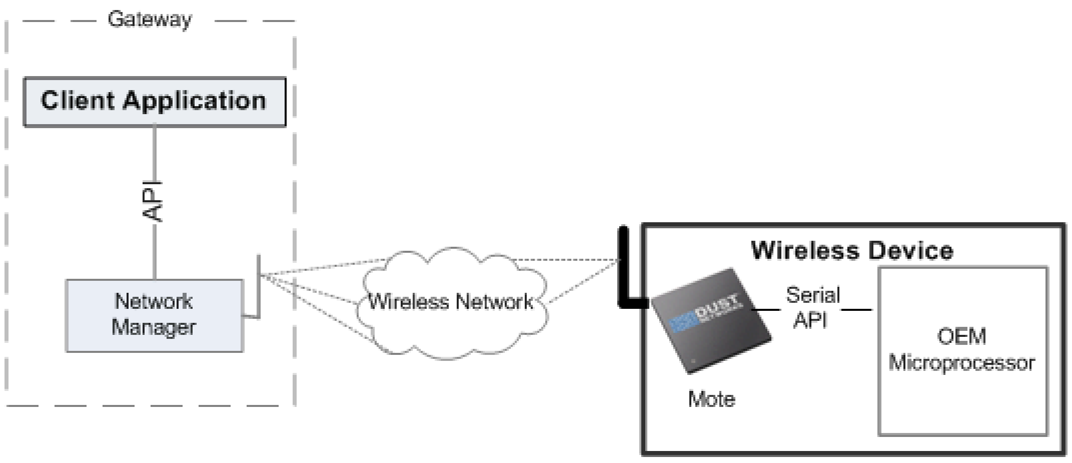
5.1 Introduction

SmartMesh WirelessHART Motes form the "body" of the network. The mote is responsible for:

- Maintaining synchronization to the network
- Forwarding data from descendants
- Generating health reports to continually update the manager's picture of the network
- Generating alarms to indicate failures, such as a lost path
- Presenting user interfaces to a sensor application

The LTC5800-WHM is a single-chip solution intended to be embedded in the customer's design. The LTP5901-WHM and LTP5902-WHM are modularly certified so they can be integrated without the need for radio certification. The LTP5900-WHM is a backwards-compatible 22-pin form factor module for using the LTC5800-WHM in legacy designs.

The following diagram illustrates the terminology used throughout this guide.



Device - Mote and OEM Microprocessor

5.1.1 Steps in a Mote Design

As with the manager, although the mote is busy with networking tasks, the sensor application has relatively few things it must do:

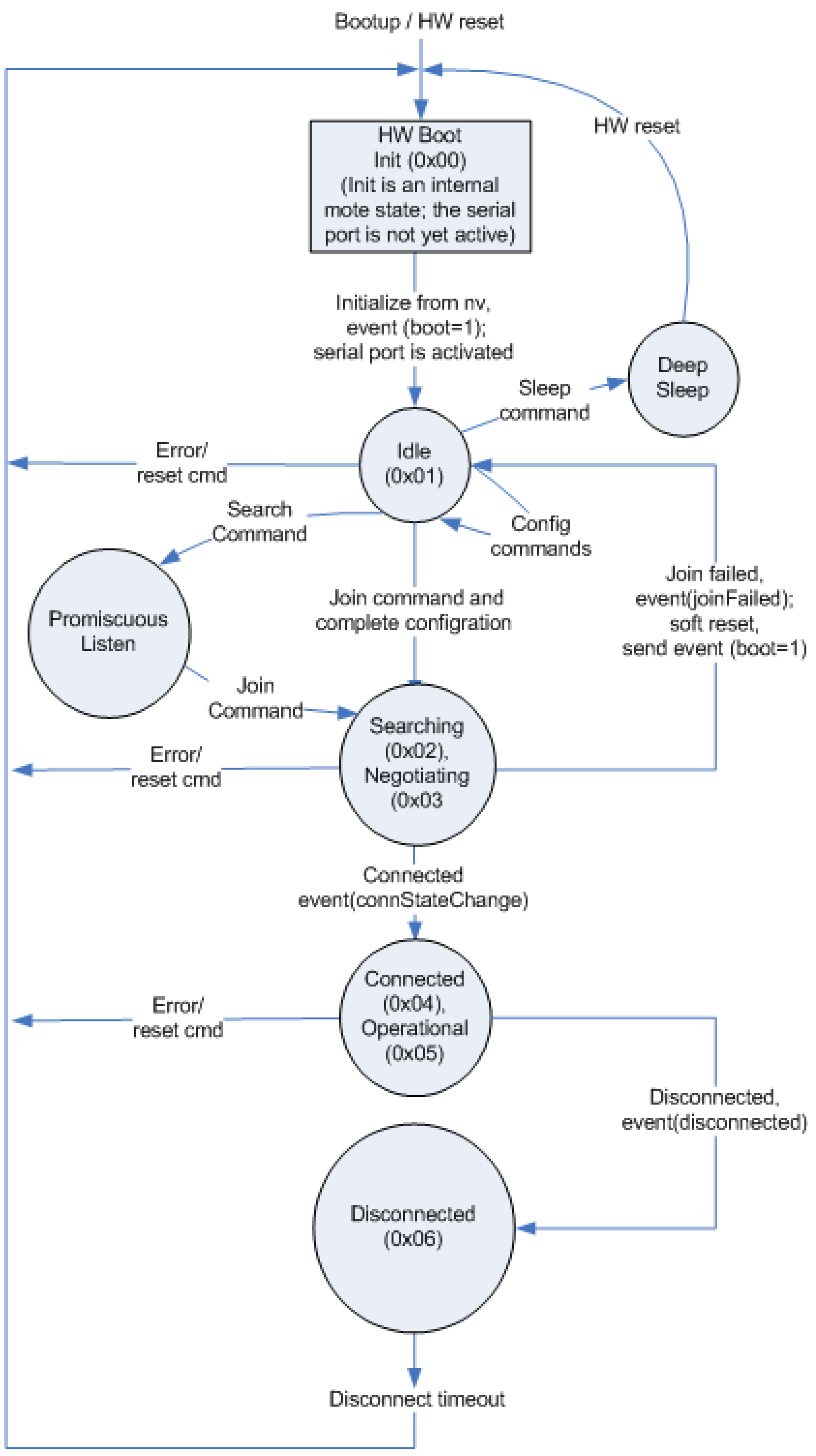
- Acknowledge the mote boot event
- Configure any parameters needed prior to join (such as *joindutycyle*)

- Use the *join* API to cause a mote to being searching for a network
- Monitor the mote state to see when it is ready to accept data
- Request services in order to publish data
- Send data and respond to application layer messages.

The SmartMesh WirelessHART Mote API Guide covers other commands to configure the mote. The SmartMesh WirelessHART Mote CLI Guide covers using the human interface to observe mote activity.

5.2 Mote State Machine

The following state machine describes the general behavior of a mote during its lifetime, and is provided for the user's information. In general an application only needs to issue the join command to enter a network, and issue a small subset of API commands to send data.


Bootup / HW reset
HW reset
HW Boot
Init (0x00)
(Init is an internal mote state; the serial port is not yet active)
Initialize from nv, event (boot=1); serial port is activated
Sleep command
Deep Sleep
Error/ reset cmd
Idle (0x01)
Search Command
Config commands
Promiscuous Listen
Join command and complete configration
Join failed, event(joinFailed); soft reset, send event (boot=1)
Join Command
Error/ reset cmd
Searching (0x02), Negotiating (0x03
Connected event(connStateChange)
Error/ reset cmd
Connected (0x04), Operational (0x05)
Disconnected, event(disconnected)
Disconnected (0x06)
Disconnect timeout

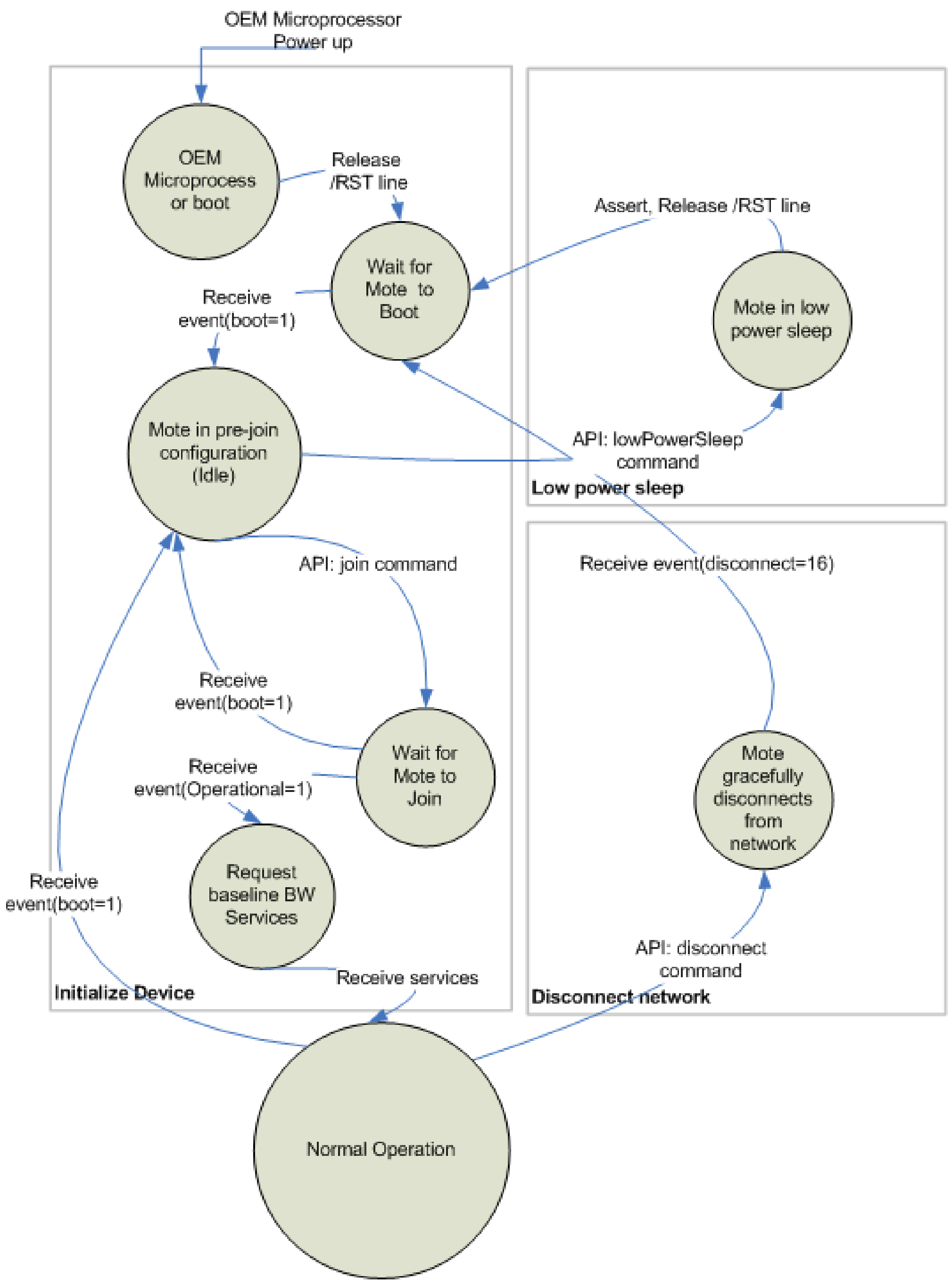
The mote states are as follows:

- **Idle:** While in this state, the mote accepts configuration commands. Upon receiving a join command, the device moves into the **Searching** state. This state is equivalent to the HART **Low Power** state.

- **Deep Sleep:** The mote enters deep sleep when it receives the *lowPowerSleep* command from the attached serial processor. In this state, the device can no longer respond to serial commands and must be reset to resume normal operation. For power consumption information, refer to the mote product datasheet.
- **Promiscuous Listen:** A special search state, invoked by the *search* command, where the mote listens for advertisements from any Network ID, and reports heard advertisements. The mote will not attempt to join any network, and will proceed to the Searching state when given the join command.
- **Searching**: In this state, the device keeps its receiver on with a configurable duty cycle while searching for the network. This state is equivalent to the HART Active Search state or Passive Search state, depending on the duty cycle setting.
- **Negotiating**: The mote has detected a network and has received a join request from the manager.
- **Connected**: The mote has joined the network and established communications with the network manager, but has no links for sending or receiving application data. This state is equivalent to the **Quarantine** state in WirelessHART.
- **Operational**: The mote has links to the network manager and the device has sufficient bandwidth for basic communication with the control program (through the network manager).
- **Disconnected**: The mote no longer has links to the network and will reset after the disconnect timeout.

5.3 Joining

The OEM microprocessor must interact with the mote in order for it to boot and connect to the network. The following sections walk through the steps required to bring a device onto the wireless mesh network, as illustrated in the following diagram.



State Machine–Connecting to Network and Disconnecting from Network

5.3.1 OEM Microprocessor Boot

The OEM microprocessor typically goes through a boot sequence after power is applied to it. The OEM microprocessor should keep the mote in reset until its serial port is fully configured and ready to receive messages from the mote. Since the mote still consumes current while powered and held in /RST state (see product datasheet for current consumption specifications), the OEM microprocessor should minimize the time between mote power up and when it releases the mote /RST line.

5.3.2 Mote Boot

When the OEM microprocessor releases the mote /RST line, the mote goes through its normal boot process. Refer to the product datasheets for the expected mote boot time. As documented in the SmartMesh WirelessHART Mote Serial API Guide, the mote sends a *boot* notification after it boots. The mote continually sends the boot notification until the OEM processor can acknowledge it. Once boot is acknowledged, the mote enters the Idle state.

5.3.3 Pre-join Configuration

When the mote is in the Idle state, the OEM microprocessor can configure various operational parameters. Some operational parameters (stored in mote non-volatile memory) need to be programmed at least once in the life of the mote while others may need to be programmed each time the mote enters the Idle state.

Program the following parameters at least once in the life of the mote:

- *setNVParameter<networkId>* and *setNVParameter<joinKey>*—For improved security, the Network ID and join key for the device should be set to values other than the Dust factory default settings.
- *setNVParameter<powerInfo>*—This parameter describes the power source for the field device, and specifically the portion of power allocated to the mote.
- *setNVParameter<HARTantennaGain>*—In WirelessHART-compliant networks, the mote uses this value to reply to HART commands 797 and 798. For more information, see CMD 797 Write Radio Power Output and CMD 798 Read Radio Power Output in the Wireless Command Specification (HCF_SPEC-155).

In addition to the required parameters described above, the OEM microprocessor may choose to change the following parameters:

- *setNVParamete<macAddress>*—This parameter can be used to assign the mote a different MAC address (for example, a HART-compliant MAC address). By default, the mote comes with a MAC address from Dust Networks' IEEE address space.

The OEM microprocessor should also take advantage of the following parameters available to it through the mote serial API:

- *setNVParameter<joinKey>*—For improved security, the join key for the device should be set to values other than the Dust factory default values. Note that the manager's Access Control List (ACL) table must be in synch with the keys programmed on the devices.
- *setNVParameter<ttl>*—Time To Live (TTL) is the maximum number of hops the packet may traverse before it is discarded from the network.

Program the following the following parameters each time the mote enters the Idle state:

- *setParameter<joinDutyCycle>*—The default setting was selected to reach a reasonable join rate without excessive power consumption. Should a faster join rate be preferred at the expense of higher current consumption (for network demonstrations, or development systems), refer to the SmartMesh WirelessHART Mote API Guide for further details and guidance for setting join duty cycle.
- *setParameter<hartDeviceInfo>*—Sets the HART CMD 0 and CMD 20 information as defined in the Universal Command Specification, (HCF_SPEC-127) for use during the join process. This only applies to WirelessHART-compliant devices.
- *setParameter<hartDeviceStatus>*—Sets the HART Device Status and HART Extended Device Status, which the mote uses when sending HART packets. Device status is described in the Command Summary Specification (HCF_SPEC-99), table 12. Extended device status is described in the Common Tables Specification (HCF_SPEC-183), table 17. This command only needs to be called once before joining - the device status and extended status are sent by the OEM processor in each user packet, and are cached by the mote for its own packets. When there is a wireless stack configuration change, the mote will generate a *configChanged* event to let the OEM processor know to update its copies of the the status bytes. See Key WirelessHART Command Support for more details. This only applies to WirelessHART-compliant devices.

Refer to the SmartMesh WirelessHART Mote API Guide for a complete list of available parameters.

5.3.4 Network Joining

Once the OEM microprocessor has completed its pre-join configuration, it may issue the *join* API command. The time for network joining varies from network to network and from mote to mote, and depends on factors such as number of available neighbors, network traffic level, and RF interference. The mote will send an *event* packet indicating that the mote is operational. At this point, the mote has joined the network, it is now capable of receiving data from the manager and setting up bandwidth services to transmit data.

> The *join* command is a Dust Networks API command and should not be confused with the HART CMD 771 Force Join Mode described later in this guide. For more information about network joining for WirelessHART applications, see CMD 771 Force Join Mode.

5.4 Services

The SmartMesh network has great flexibility to control bandwidth allocation, allowing it to accommodate a wide variety of applications. The network can be configured for simple data reporting applications with a single parameter at the manager API, or alternatively, the network can use dynamic bandwidth services (now called *Timetables* in WirelessHART 7.4) that allow it to allocate different bandwidth to individual devices and change bandwidth allocation over time. This chapter describes how to use the dynamic bandwidth model for full flexibility.

5.4.1 Service Characteristics and Timing Parameters

The dynamic bandwidth services mechanism provides a means by which the mote may request bandwidth from the network manager. The following table describes the characteristic usage for bandwidth services.

Application Domain	**Characteristics**	**Bandwidth Originator**
Maintenance	Symmetric data traffic, such as request/response pairs.	Network manager
Publish	Data sent regularly from the device to the client application. For example, send sensor reading once every 5 minutes.	Device (upon receipt of request to burst)
Block transfer	Temporary high-speed transfer of large chunks of data. For example, uploading a diagnostics file from mote to client application.	Upstream - Device (upon receipt of request) Downstream - Client application
Event	Latency-sensitive data sent infrequently from mote to client application. For example, process alarms.	Device (during initialization)

The following table contains timing specifications for service requests and retries for Publish, Block Transfer, and Event services. Maintenance service is not subject to service rejection because it is automatically allocated by the manager at join time.

Variable	**Description**	**Min**	**Max**	**Units**
T_{Svc_Retry}	Time from the OEM microprocessor receiving a service rejection (via *serviceIndication* mote API) to OEM microprocessor retrying the service	2		Minutes

5.4.2 Service Types

SmartMesh WirelessHART motes support bandwidth services to accommodate the dynamic bandwidth needs of the complex applications common in monitoring and control. The services mechanism provides a means by which the mote may request bandwidth from the network manager. When used in conjunction with a SmartMesh WirelessHART manager, the device receives the full benefits of bandwidth services to support the range of tasks that a fully productized device needs to perform—not just regular data reporting, but also rapid request-response configuration, block transfers, and alarm messages.

Services may be either of the following:

- Manager-originated-Bandwidth services pushed from the network manager to the device.
- Device-originated-Bandwidth services requested by the OEM microprocessor via the API commands, as described below.

5.4.3 Non-Service Bandwidth Control

The mote may skip request services only if all of the following conditions are met:

- The system is not WirelessHART compliant

- Dynamic bandwidth allocation is not required.
- The client application (communicating to the SmartMesh WirelessHART manager) uses the bandwidth control manager API commands (e.g. *setConfig Network requestedBasePkPeriod or activateFastPipe)*

to ensure bandwidth is always sufficient for the field device.

If the above conditions are met, then the OEM microprocessor is not required to request services and may use service ID 0x80 when sending data via the send command.

5.4.4 Using the Services API

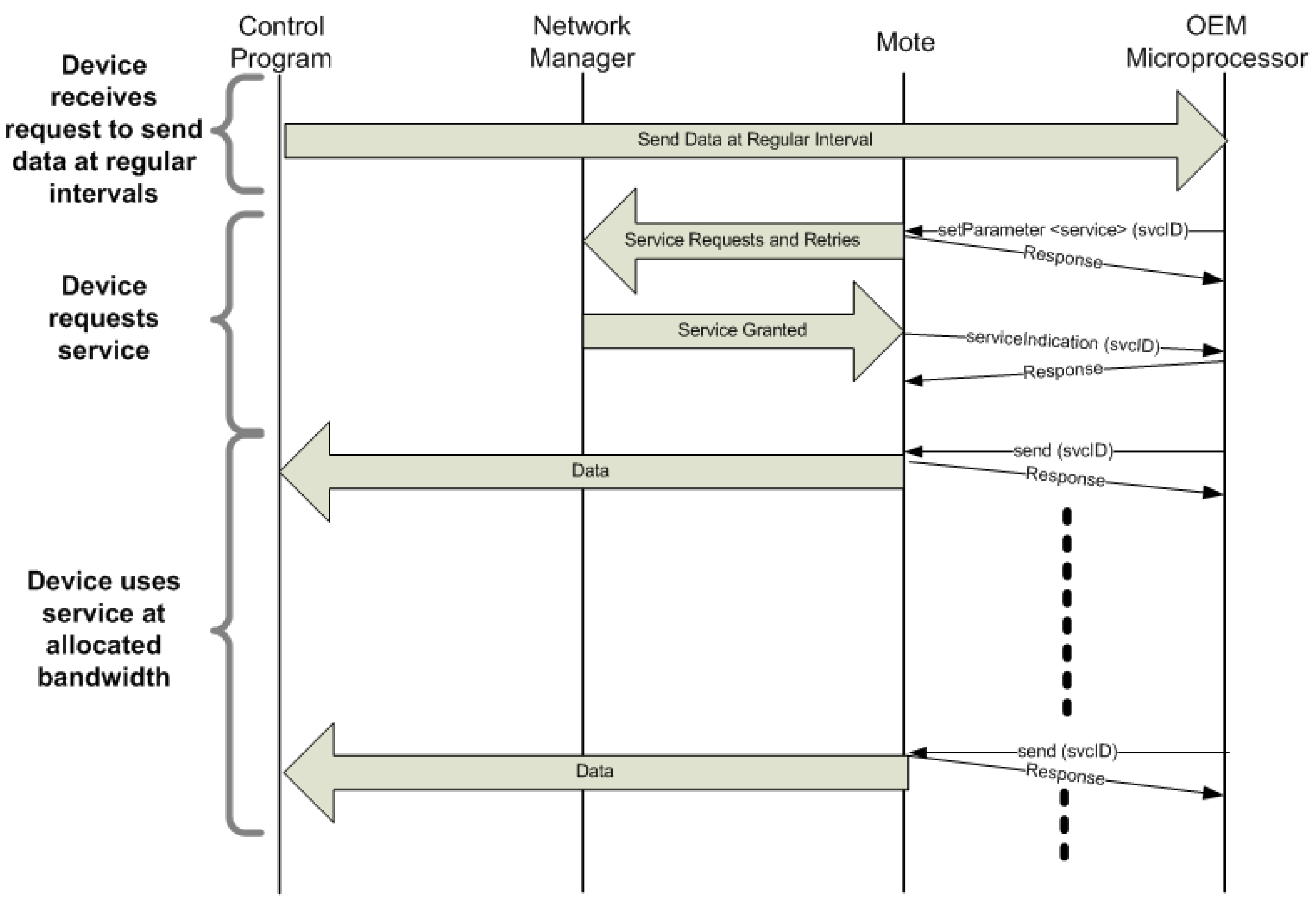
Service-related API Commands

For device-originated services, the mote makes the service requests based upon the values of the service table entries that are filled in by the OEM microprocessor via the services API. The mote manages the service request packet exchange with the network manager (initial request, retries, and delayed response handling). The mote maintains a service table where each entry describes the service and the current service status (for example, whether the service is active).

In the 7.4 version of the HART specification, the term *service* has been changed to *timetable*. API calls still use the "service" term.

The mote serial API provides the following commands to manage services:

- *setParameter<service>*—This command enables a microprocessor to request a new service or modify an existing device-originated service. A successful response to this command indicates that the mote has updated its service table and will send a service request to the network manager once the mote is operational.
- *serviceIndicatio*n—The mote sends this command to the microprocessor when it updates its service table after receiving notice of the following:
  - A new device-originated service was granted as a result of service request initiated by the microprocessor
  - A manager-originated service was created
  - An existing service was updated or deleted
- *getParameter<service>*—This command enables the microprocessor to check the services description and status of service table entries. The service table contains both manager-originated and device-originated services.



Control Program
Network Manager
Mote
OEM Microprocessor
Device receives request to send data at regular intervals
Device requests service
Device uses service at allocated bandwidth
Send Data at Regular Interval
Service Requests and Retries
setParameter <service> (svcID)
Response
Service Granted
serviceIndication (svcID)
Response
Data
send (svcID)
Response
Data
send (svcID)
Response

Transaction Diagram—Bandwidth Service Requests

The following are specific usage scenarios for requesting services.

- **Retrying service requests**—The response to a service request may take some time if it results in the network manager adding bandwidth to the network. The mote automatically handles communication retries with the network manager until it receives a response. (In WirelessHART terminology, the mote will handle Delayed Response.) If a service is denied, the microprocessor may re-send a service request to the network manager by reissuing the *setParameter<service>* request. A service may be denied if there is a problem in the network (for example, insufficient bandwidth upstream of the mote). The microprocessor may retry the service request after waiting two minutes to allow the network manager time to attempt to resolve the network problem.
- **Modifying an existing service**—If a service is active at rate X, and the microprocessor requests an update to a rate Y, the service state remains **Active**, but the service rate remains at rate X until the network manager responds to grant rate Y. The mote forwards a *serviceIndication* event to the microprocessor when it receives a notification from the network manager.
- **Deleting a service**—To delete a service, the microprocessor should issue a *setParameter<service>* command with a time value of zero. For more information, refer to the *setParameter<service>* command in the SmartMesh WirelessHART Mote API Guide.

- **Network manager removes a service**—Dust network managers continuously optimize network links to maintain network performance in the face of RF challenges. However, in the event that the manager cannot sustain a service due to network conditions, it will reduce the allocated bandwidth to a level that effectively disables the service. The mote will receive notification from the network manager and forward a *serviceIndication* command to the microprocessor.

The microprocessor may choose to submit another request to update the service at a future time.

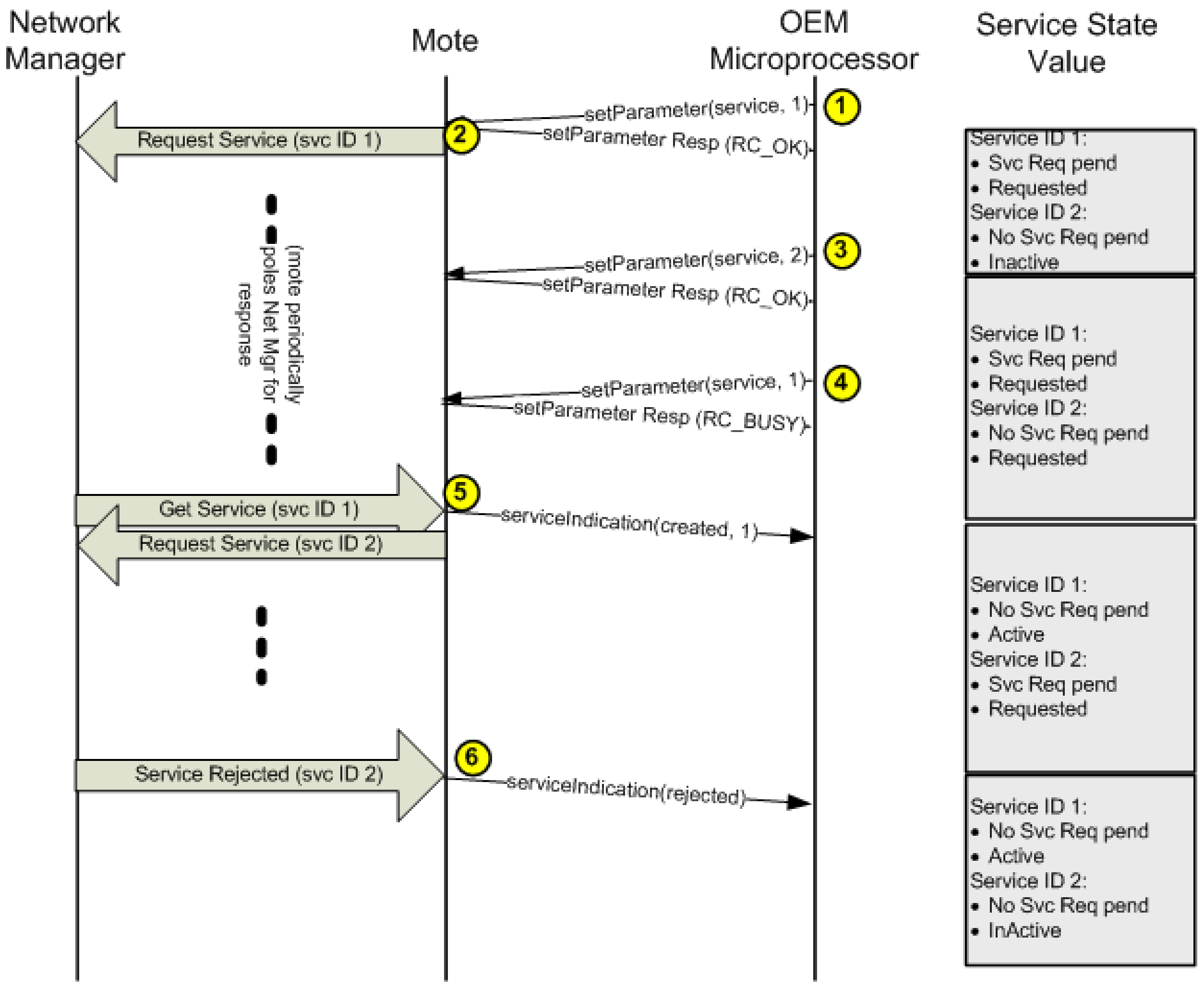
Service Table

The mote service table describes both granted and requested services. The parameters below are used in the *setParameter<service>* and *getParameter<service>* commands.

- **Service ID**—A unique identifier for the service.
    - **Service IDs from 0x00 to 0x7F**—Indicate device-originated services that the microprocessor has requested.
    - **Service IDs from 0x80 to 0xFF**—Indicate manager-originated services.
- **Service State**—Used in *getParameter<service>*, this field reflects the state of the service. The fields are as follows:
    - **Status bit**—Indicates whether a service is inactive, active, or requested. Here, “requested” means OEM microprocessor has sent requested a service through *setParameter<service>*, but either the mote has not yet sent a request to the network manager, or the network manager has not accepted/denied.
    - **Service pending bit**—Used for services in the requested state, the servicepending bit indicates if the mote has sent a service request to the network manager.
- **Service Flags**
    - **Source**—The mote is the source/sender of data.
    - **Sink**—The mote is the receiver of data.
    - **Intermittent/regular interval**—Designates whether the traffic will be used at regular intervals, or on an intermittent basis (for example, events).
- **Application Domain**
    - **Maintenance**—Symmetric data traffic, such as request-response pairs.
    - **Publish**—Data sent at a regular interval from mote to the gateway.
    - **Event**—Data sent infrequently from the mote to gateway (such as process alarms).
    - **Block transfer**—Temporary high-speed transfer of large chunks of data.
- **Destination Address**—Because services are always between the gateway and the mote in SmartMesh WirelessHART network, the destination address should be specified as 0xF981.
- **Time**—The time variable describes the rate of data.
    - If data is being regularly reported (the intermittent bit is not set on service request flag), then time is the period between data sent in milliseconds.
    - If the data is intermittent, then time is the desired latency in milliseconds.
    - To delete a service, set the time field of the desired service to zero. Service request flags, application domain, and destination address are ignored by the mote when time equals zero.

The following transaction diagram further illustrates how the service state byte is updated during both a successful and an unsuccessful service request.

1. The OEM Microprocessor initiates a service request by calling *setParameter<service>* on an unused service ID.
2. The mote receives the *setParameter<service>* command, updates its service table, and initiates a wireless request to the network manager. The mote handles all retries until it receives a success, warning, or error (in WirelessHART terminology, this is a non-DR response).
3. The OEM microprocessor requests a service for service ID 2.
4. The OEM microprocessor attempts to request a service ID 1. However, since there is a service request pending already for service ID 1, a serial API error is returned.
5. The network manager grants the service. The mote updates its service table, and sends a *serviceIndication* packet to the OEM Microprocessor. The mote then initiates a request to the network manager for the next valid service in the service table.
6. The network manager rejects the service. The mote updates its service table, and sends a *serviceIndication* packet to the OEM microprocessor.



Transaction Diagram—Service State Values

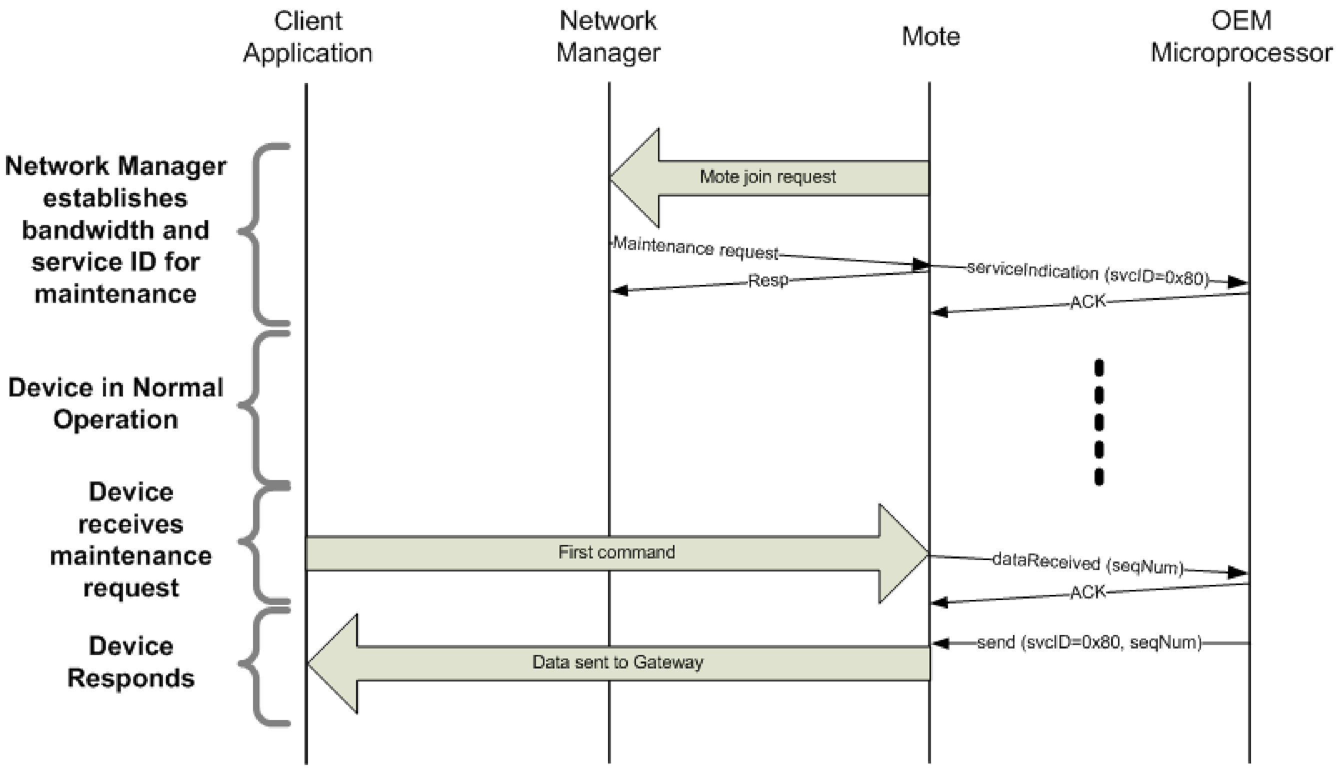
5.4.5 Maintenance

The purpose of the Maintenance service is to give the wireless network a minimum overhead bandwidth for basic network control communications. The Maintenance service can also be used for user data when all devices in the network require relatively the same amount of bandwidth. Unlike the other bandwidth services, the Maintenance service is not designed to accommodate individual device bandwidth requirements, which may vary widely from device to device

Maintenance Service Origination

When a mote joins the network, a network manager will establish Maintenance bandwidth and push the service ID to the mote. Maintenance is therefore a manager-originated service.

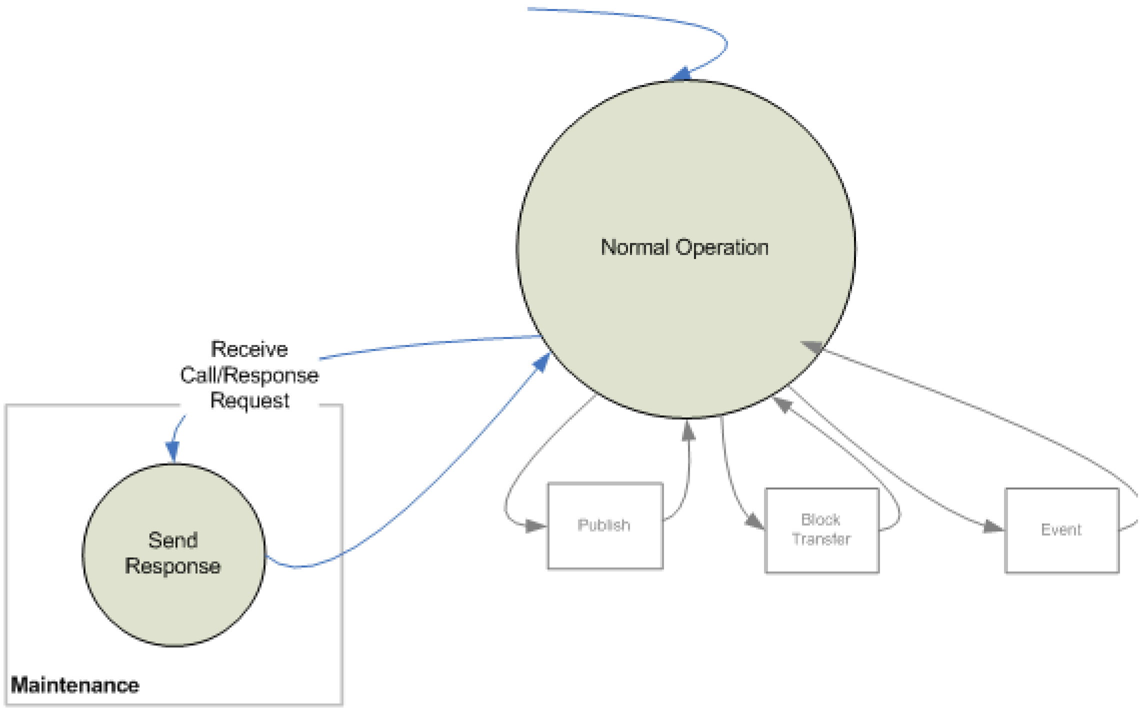
The network manager will write the service to the mote. Upon receipt of the write service from the network manager, the mote will pass a *serviceIndication* mote serial API message to the OEM microprocessor. This message includes the service ID and application domain among other information.


Client Application
Network Manager
Mote
OEM Microprocessor
Network Manager establishes bandwidth and service ID for maintenance
Mote join request
Maintenance request
serviceIndication (svcID=0x80)
Resp
ACK
Device in Normal Operation
Device receives maintenance request
First command
dataReceived (seqNum)
ACK
Device Responds
send (svcID=0x80, seqNum)
Data sent to Gateway

Maintenance–Single Request/Response Pair Transaction Diagram

Using Maintenance Service

The following state machine diagram illustrates how an OEM microprocessor may respond to maintenance requests. As described above, the Maintenance bandwidth and service ID are established by the network manager. Therefore, the device (more specifically the OEM microprocessor) may be assured that it will have the Maintenance service ID prior to receiving a Maintenance request. The result is the following simple state machine showing that when the OEM microprocessor receives a Maintenance request, it may reply to it using the Maintenance service ID as soon as the request is received.



State Machine–Maintenance Application Domain

5.4.6 Publish

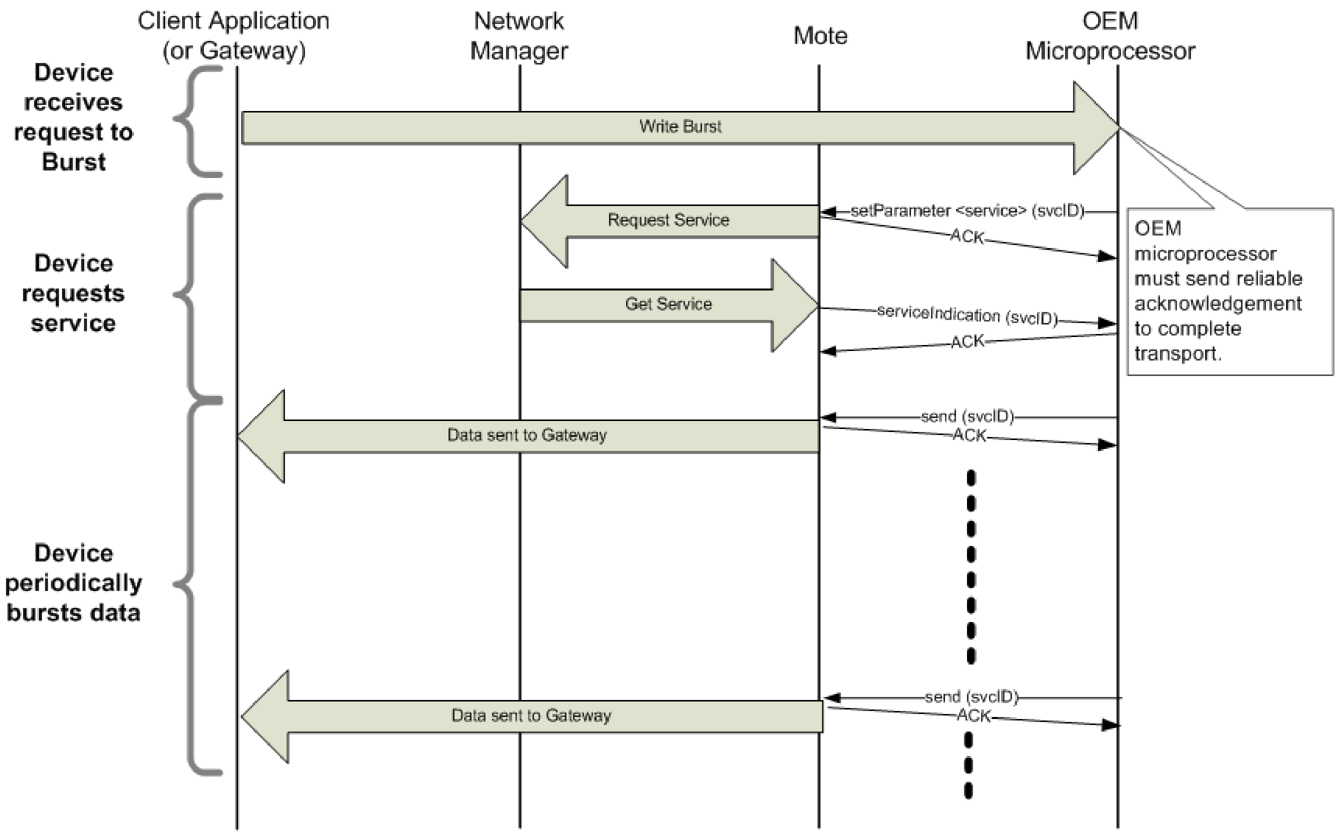
The Publish service is enabled when the device needs to regularly send data, such as reporting a sensor reading on a fixed interval. Once a device is configured, it may then publish data indefinitely (potentially for years). The device should continue to publish data until it is instructed to stop. Similarly, once a Publish bandwidth service is established, it is expected to remain in use until it is overwritten by a new Publish command.

Publish Service Origination

In the Publish application domain, the device is responsible for requesting bandwidth. The transaction diagram below provides an overview of bandwidth service handling for Publish.

When the OEM microprocessor receives a command to burst data, the OEM microprocessor must request a bandwidth service via the *setParameter<service>* mote serial API command. Once the bandwidth service has been granted, the OEM microprocessor may begin publishing data at a rate that does not exceed the granted bandwidth.

In WirelessHART-compliant applications, the OEM microprocessor can receive HART common practice commands to burst data. The OEM microprocessor is responsible for managing command responses, such as Delayed Response (DR) responses to the HART Gateway. (Refer to the HART specifications HCF_SPEC-290, for more details on DR).



Client Application (or Gateway)
Network Manager
Mote
OEM Microprocessor
Device receives request to Burst
Write Burst
Device requests service
Request Service
setParameter <service> (svcID)
ACK
Get Service
serviceIndication (svcID)
ACK
OEM microprocessor must send reliable acknowledgement to complete transport.
Device periodically bursts data
Data sent to Gateway
send (svcID)
ACK
Data sent to Gateway
send (svcID)
ACK

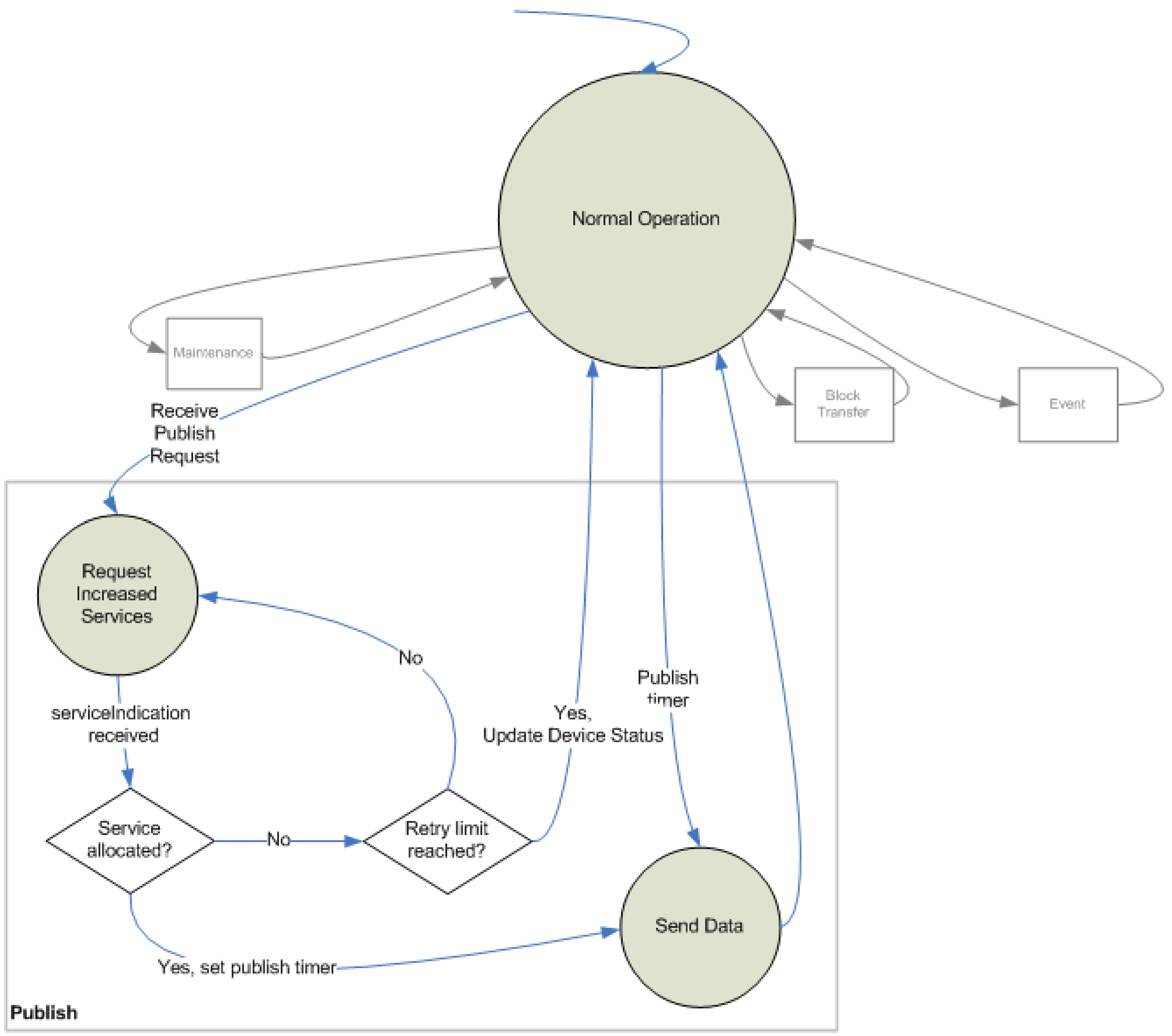
Publish–Transaction Diagram

Using the Publish Service

The following state machine diagram illustrates a typical OEM microprocessor design for handling service requests specifically for the Publish application domain. When the OEM microprocessor receives a request to publish data, it should request a bandwidth service and then wait for the network manager response, as indicated by a *serviceIndication* packet from the mote serial API. Note that because the service request exchange between the mote and the network manager uses reliable transport, the OEM microprocessor will always either receive a *serviceIndication* API notification or in the case of transport failure, the mote will reset.

- If the *serviceIndicatio*n response is received, but the service is either rejected, or the granted bandwidth is not sufficient to service the burst request, then the OEM microprocessor should retry the service request after waiting a period of T_{Svc_Retry} as defined above in Service Characteristics and Timing Parameters. Even in a well-managed network with good path stability, the network manager may take many minutes to add the bandwidth needed to satisfy the service request.
- If the service is granted successfully, the OEM microprocessor may begin bursting data at regular intervals (consistent with the level of bandwidth service).

In WirelessHART applications, after several unsuccessful retries (for example, 5 retries), the OEM microprocessor must set the “Capacity Denied” status and the “More status available” bits in the Device Status byte. The OEM microprocessor must call *setParameter<hartdeviceStatus>* with the updated value.



State Machine–Publish Application Domain

setParameter <service> Parameter	Value	Description
Service Index	<Unused device-originated service ID>	
Service Request Flags	0x01	Device is source, but not sink, not intermittent
Application Domain	0x00	Publish
Destination address	0xF981	
Time		Set to data packet interval

Service Request Parameters for Publish

Send Parameter	Value	Description
Flags	Bit 7 = 0 Bit 6 = 0 Bits 5-3 = 0 Bit 2 = 0 Bit 1 = <packet ID>	HDLC request, Best effort, with packet ID enabled. For details on packet ID, see the section titled "Flags" under the HDLC packet format discussion in the SmartMesh WirelessHART Mote Serial API Guide.
Destination address	0xF981	
Service Index	<service ID>	Same as what was used in service request
Application Domain	0x00	Publish
Priority	0x02	For WirelessHART applications, this corresponds to "data" priority per Table 18, HCF_SPEC-085.
Reserved	0xFFFF	
Sequence number	0xFF	

Send Parameters for Publish

Handling Reductions in Available Publish Bandwidth

At times the network manager may update an existing service, such as during bandwidth restrictions. When the mote receives this service update from the manager, it will forward a *serviceIndication* packet to the OEM microprocessor. The OEM microprocessor must include logic to handle reductions in available bandwidth for Publishes.

5.4.7 Block Transfer

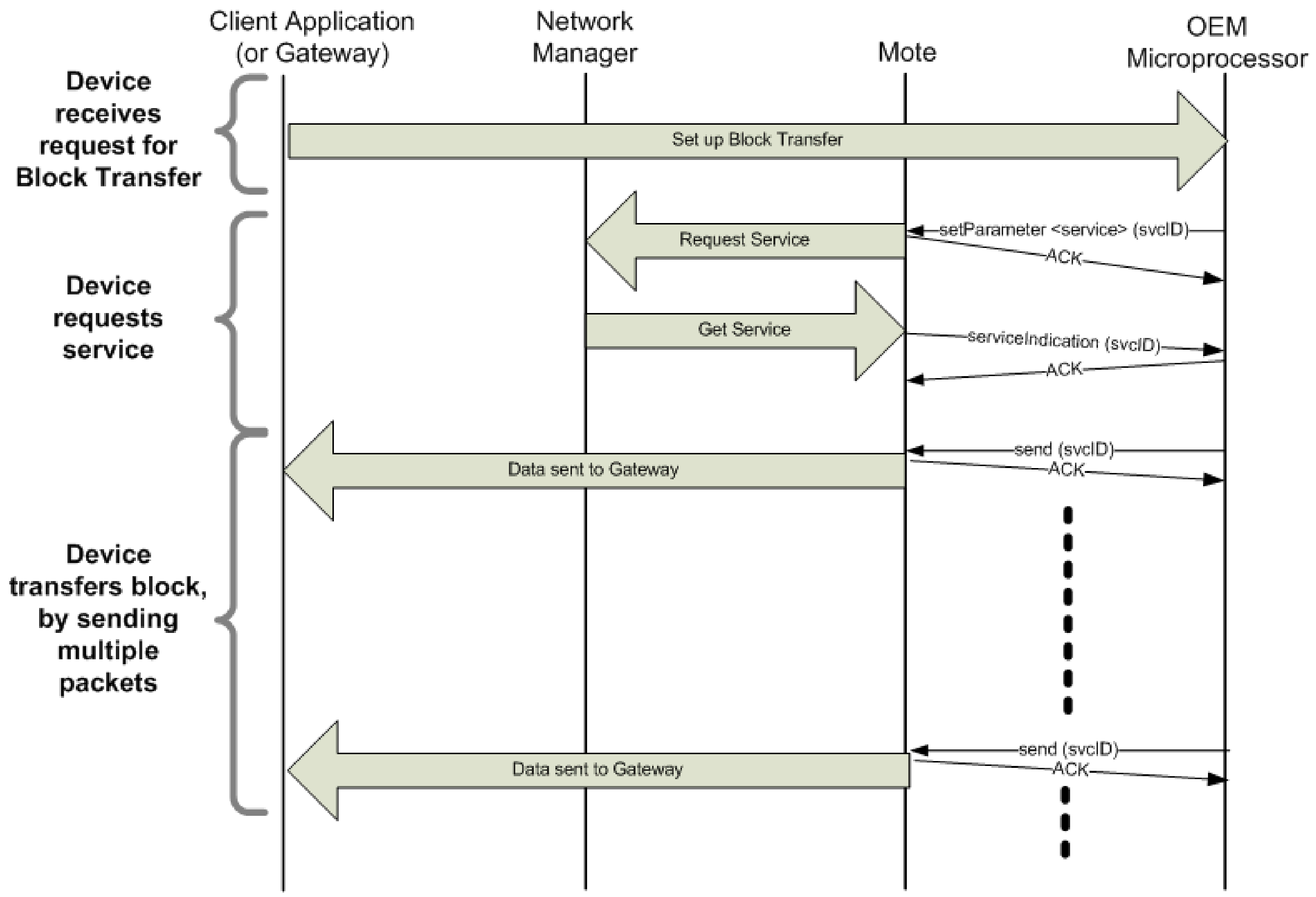
The Block Transfer application domain is used for sending large consecutive blocks of data, such as data log files. In a SmartMesh WirelessHART network, although block transfers can occur in both upstream and downstream directions, the network manager (or HART gateway) handles bandwidth allocation for the downstream block transfers. Therefore, the OEM microprocessor may receive downstream block transfers without any special handling. In contrast, since the device is the sender in an upstream block transfer, the device requests and owns the bandwidth service. The following sections cover upstream block transfers.

Block Transfer Service Origination

By definition, block transfers have a beginning and end, therefore the bandwidth service for a block transfer should be requested when the transfer is requested and deleted or reduced when the transfer is complete. The following transaction diagram provides an overview of the Block Transfer service request.

When the OEM microprocessor receives a command to set up a block transfer, the OEM microprocessor must request the bandwidth through the mote *setParameter<service>*command. Once the bandwidth service has been granted, the OEM microprocessor may begin sending data at a rate not to exceed the granted bandwidth. Once the bandwidth service has been granted, the OEM microprocessor may begin sending data at a rate not to exceed the granted bandwidth.

In WirelessHART-compliant applications, a HART command is used to set up a block transfer. The OEM microprocessor is responsible for managing the HART Command responses, such as Delayed Response (DR) responses to the HART Gateway.



Send Block Transfer–Transaction Diagram

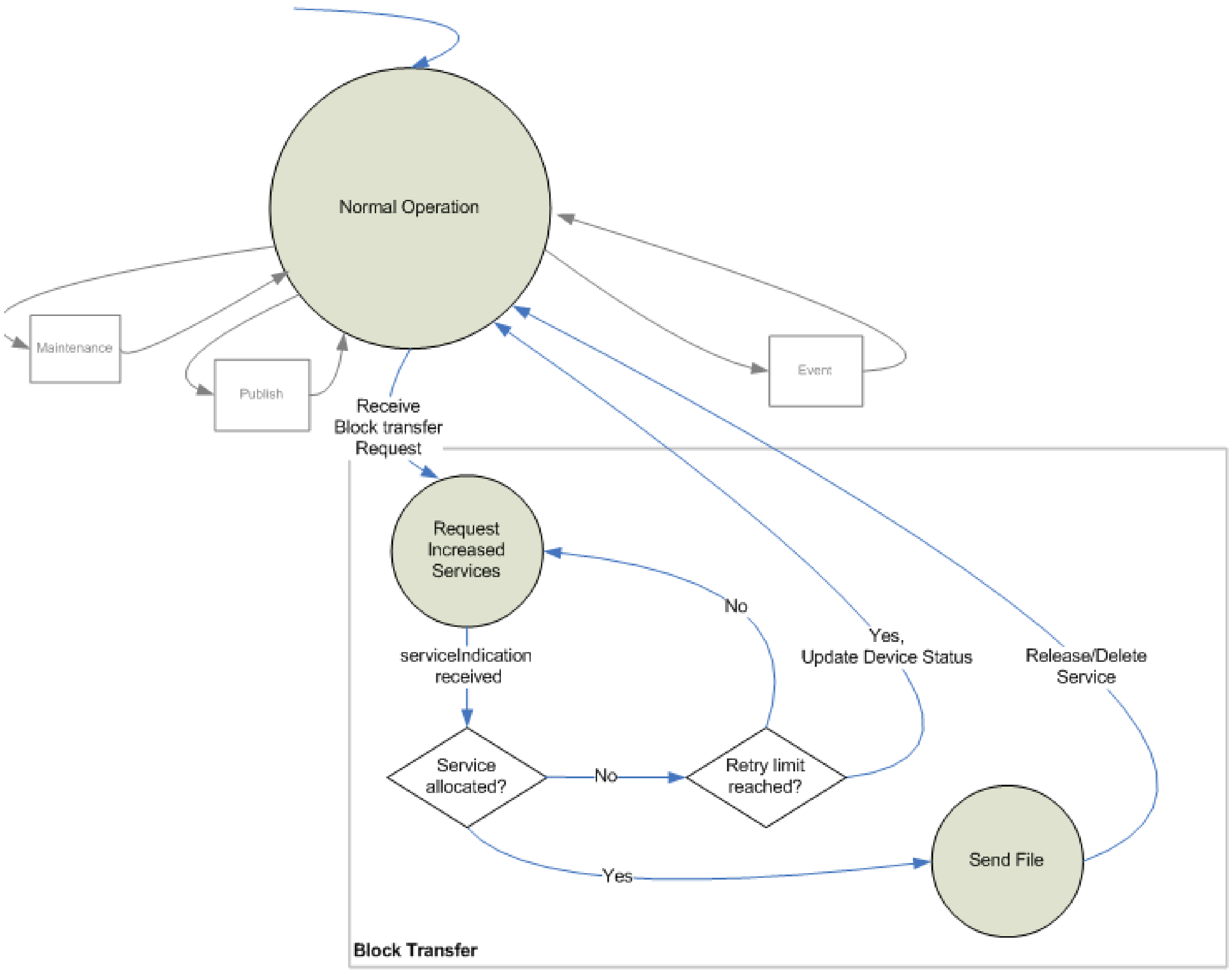
Using the Block Transfer Service

The following state machine diagram illustrates a typical OEM microprocessor design for handling service requests specifically for the Block Transfer application domain. When the OEM microprocessor receives a request for a block transfer, it should request a bandwidth service, then wait for the network manager response, as indicated by a *serviceIndication* packet from the mote serial API. Note that because the service request exchange between the mote and the network manager uses reliable transport, the OEM microprocessor will always receive a *serviceIndication* API notification.

- If the *serviceIndication* response is received, but the service is rejected, then the OEM microprocessor should retry the service request after waiting a period of T_{Svc_Retry} as defined in Service Characteristics and Timing Parameters . Even in a well-managed network with good path stability, the network manager may take many minutes to add in the bandwidth needed to satisfy the service request.
- If the service is granted successfully or at a reduced rate, the OEM microprocessor may begin bursting data at regular intervals (consistent with the level of bandwidth service). If the sending device exceeds granted services, there is a risk of congesting the network.

- After the OEM microprocessor is finished sending data, it must request that the Block Transfer service be deleted. Only one Block Transfer service can be active at a time, and if the OEM microprocessor does not delete the service it will be unavailable for other devices to use.

In WirelessHART-compliant applications, after several unsuccessful retries (for example, 5 retries), the OEM microprocessor must set the “Block Transfer Pending” status and the “More status available” bits in the Device Status byte. The OEM microprocessor must call *setParameter<hartdeviceStatus>* with the updated value.



State Machine–Block Transfer Application Domain

5.4.8 Events

The Event bandwidth service is used for sending data packets during exceptions, such as warnings. While these events normally occur infrequently, when they do occur, usually delivery of the data packet is urgent. Therefore, the bandwidth services must be established ahead of time. Note that because the service requests exchanged between the mote and the network manager use reliable transport, the OEM microprocessor will always either receive a *serviceIndication* API command or in the case of transport failure, the mote will reset.

Event Service Origination

In the Event bandwidth service, the device is responsible for requesting bandwidth. The following transaction diagram provides an overview of bandwidth service handling for Events.

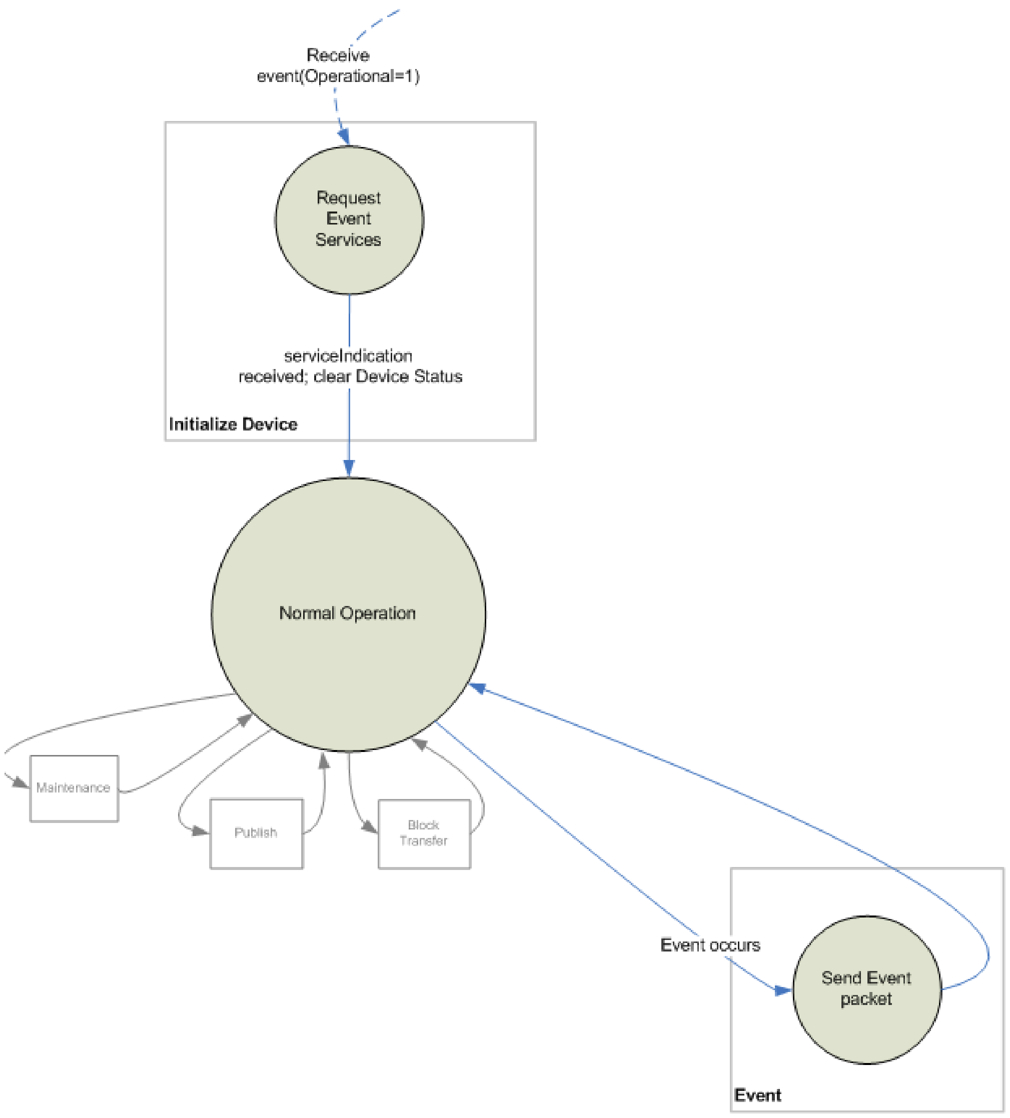
As described in Connecting and Disconnecting, after starting up the mote the OEM microprocessor goes through a sequence of configuration commands to initialize the device. Once the mote has joined the network and is in the Operational mote state, the OEM microprocessor should request an event service through the *setParameter<service>* command. One of the parameters of the service request is the requested latency for packets. In general, faster latency comes at the cost of higher power consumption. SmartMesh WirelessHART network managers will balance latency against power consumption for event services.

Current generation managers do NOT lay in additional bandwidth in response to an Event service request - the existing Maintenance service bandwidth is used. An OEM processor must gracefully tolerate a *serviceIndication* with a lower value than requested.

However, the OEM microprocessor should request latency reasonably, should the device be used with non-Dust Networks managers. After the service request, the OEM microprocessor should then wait for the network manager response, as indicated by a *serviceIndication* packet from the mote serial API. Note that because the service requests exchange between the mote and the network manager use reliable transport, the OEM microprocessor will always either receive a *serviceIndication* API command or in the case of transport failure, the mote will reset.

- If the *serviceIndication* response is received, but the service is rejected, then the OEM microprocessor should retry the service request after waiting a period of T_{Svc_Retry} as defined in Service Characteristics and Timing Parameters. Even in a well-managed network with good path stability, the network manager may take many minutes to add in the bandwidth needed to satisfy the service request.
- If the service has been granted successfully, the OEM microprocessor may proceed to the Normal Operation state. Later, if and when an event occurs on the device, the OEM microprocessor should use the event service ID to send its packet.

In WirelessHART-compliant applications, after several unsuccessful retries (for example, 5 retries), the OEM microprocessor must set the “Capacity Denied” status and the “More status available” bits in the Device Status byte. The OEM microprocessor must call *setParameter<hartdeviceStatus>* with the updated value. The OEM microprocessor may then proceed to the Normal Operation state.



State Machine–Event Application Domain

5.5 Communication

For serial transmissions from the microprocessor to the network, users may choose between best-effort and reliable transport types. In WirelessHART-compliant applications, the transport type is generally dictated for each message type.

5.5.1 Best-Effort Communication

Packets requiring no explicit receipt acknowledgement may be sent through the network using a best-effort communication mechanism. This lowest overhead method is best suited when no application acknowledgement is required or when a small percentage of lost packets is tolerable. At the receiver, all packets with the best-effort delivery flag are forwarded to the application layer and no acknowledgements are generated. Consequently, the sender receives no feedback about the success of individual packet delivery. Applications requiring guaranteed delivery should use the reliable communication mechanism.

Data received from the network is passed to the microprocessor by the *dataReceived* mote serial API command with the flags byte set to best effort. The microprocessor should always use best effort communication when it has a data packet to send through the network (as opposed to a response packet to a reliable request). For example, best-effort communication is recommended when periodically sending or bursting sensor data. The microprocessor sends a best-effort communication by using the *send* command with the flags byte set to best effort.

5.5.2 Reliable Communication

In reliable communication, packets are acknowledged end-to-end, providing confirmation to the application layer that the packet was successfully delivered. If the confirmation is not received, the sender will retransmit the packet. Downstream communication refers to packets sent from the controller to the microprocessor. Upstream communication refers to packets sent from the microprocessor to the controller (see the diagram below). In the case of downstream reliable communication, the microprocessor must respond via the *send* command. A response from the microprocessor is required after receiving a *dataReceived* command with the flags byte set to reliable. If the application layer has nothing to send to the mote, the payload should contain no data. Failure to send a response may result in the network manager taking corrective action, such as disconnecting the mote from the network.

Microprocessor-initiated reliable upstream is not supported.

Downstream Reliable Retries

Downstream reliable retries are handled in the following manner:

1. When the mote receives a network packet, it forwards the payload, along with the transport sequence number via the *dataReceived* serial API command. The microprocessor can compare the sequence number with that of the previous packets to check if it is a duplicate.
2. If the mote receives a duplicate packet, but has not received a send command in response, it forwards the message to the microprocessor.
3. The sensor processor replies via the *send* serial API command and includes the sequence number and source address from the request along with the payload. The mote receives this and passes it on to the network.
4. If the mote still has the buffered response from the microprocessor, it discards the duplicate and replies to the network on behalf of the microprocessor to close out the reliable acknowledgement.
5. The mote will retain the buffered response until it receives a new data packet from the manager (with new sequence number), which signifies to the mote that the manager received the response to its previous packet.

5.6 Events and Alarms

The mote serial API includes events and alarms that allow a microprocessor to have full visibility of mote states and conditions.

An alarm is an ongoing condition, such as low supply voltage or an error in non-volatile memory. Refer to the section on *getParameter<moteStatus>* in the SmartMesh WirelessHART Mote API Guide for details on reading alarms and information about the available alarms.

By contrast, an event is defined as a discrete occurrence in mote or network operation. Examples of events include a mote startup, a mote failure to join the network, or a change in alarm condition, such as an alarm opening or closing.

Users can control which events are pushed to the microprocessor by using the *setParameter<eventMask>* command.

5.7 Timestamps

Many of the benefits of a SmartMesh WirelessHART network, such as its high reliability and low power consumption, are attributable to the fact that it is a time synchronized mesh network. This means that under normal conditions every mote in the network has a shared sense of time accurate to better than a millisecond. Another advantage of time synchronization is that this sense of time is available to the microprocessor for application usage by means of the *getParameter<time>* command, which returns network time and Universal Time Clock (UTC) time. For detailed information on timestamp performance, see Application Note: Obtaining Accurate Timestamps.

5.8 WirelessHART-Compliant Applications

The information in this section is only important for OEMs developing a WirelessHART-compliant device or network. If you do not have WirelessHART compliance requirements, this section can be ignored.

5.8.1 Command Termination

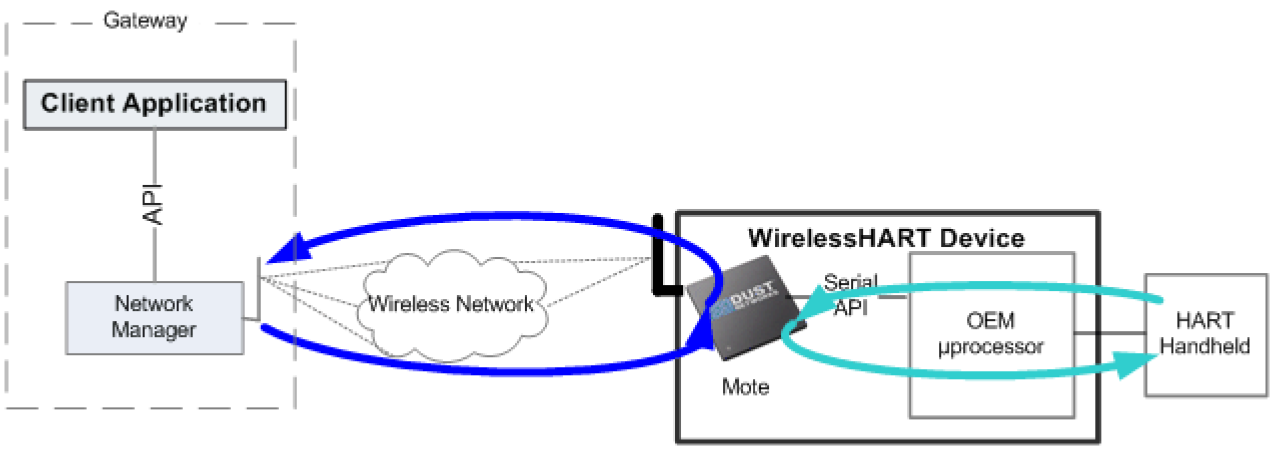
A mote provides the wireless connectivity to the mesh network and terminates a portion of the WirelessHART command set. The HART commands required of a field device must be terminated by either the mote or the microprocessor. In general, the mote terminates HART commands associated with the wireless command specification (HCF_SPEC-155) that are appropriate to a field device. The OEM microprocessor is responsible for terminating all other HART commands. The following table describes HART command number partitions. See HCF_SPEC-099 for the latest command information.

HART CMD Number	Types	Terminated By

0-767	Universal, Expansion Flag, Common Practice, Non-Public, Device-Specific, Reserved	OEM microprocessor
768-1023	**WirelessHART (HCF_Spec-155)**	**Device Commands terminated by mote**
1024-64,511	Device Family, Reserved	OEM microprocessor
64,512 - 64,765	**Wireless Device-Specific**	**Device Commands terminated by mote**
64,766-65,535	Additional Device-Specific, Reserved	OEM microprocessor

Mote-terminated Commands

The mote terminates the wireless command set (specified in HCF_SPEC-155) that are appropriate for a field device, as opposed to a network manager or gateway. Most of the HART commands terminated by the mote are related to wireless operation of the field device. For wireless HART commands that relate to operation of the field device as a whole, the mote may require information from the OEM microprocessor in order to implement the command.



Sources of Mote-terminated HART Commands

The mote can receive HART commands from two different sources:

- Over wireless from the manager, either via the gateway session or network manager session
  or
- Via the mote serial API (*hartPayload* serial API command) from the microprocessor (the command originated from HART handheld and was passed to mote by microprocessor) Messages received from the network manager typically have full access to the HART commands. Messages received via the serial API typically have full read access, but limited write or configuration access (see the Serial API Access column in the table in HCF_SPEC-155 Commands). The following table summarizes the mote's error response to commands that it receives.

Refer to HCF_SPEC-155 Commands for a complete list of commands that a mote will terminate.

Command Received By Mote	Example	Mote Response
Unimplemented Commands in HCF_SPEC-155	CMD 775, Write Network Tag	The mote responds with a valid HART response containing the HART response code 64, "Not Implemented."
Reserved WirelessHART commands (commands in the wireless address space, but not defined in HCF_SPEC-155)	CMD 64,535	The mote responds with a valid HART response containing the HART response code 64, "Not Implemented."

HART command terminated by the mote, but with access restricted to the manager (for example, from the Gateway or via the *hartPayload* mote serial API)	CMD 795, Write Timer Interval	The mote responds with a valid HART response containing the HART response code 16, "Access Restricted."
HART command not terminated by the mote (commands not in HCF_SPEC-155)	CMD 1, Read Primary Variable	If received wirelessly, passed to OEM microprocessor via *dataReceived* notification. If via serial *hartPayload* API, the mote responds with the response code = "Invalid Value" (see SmartMesh WirelessHART Mote Serial API Guide).

HCF_SPEC-155 Commands

Termination	
mt	Terminated by the mote (Device)
mgr	Terminated by the network manager
Access	
ser	Command will return valid response (non RC16) when requested via serial API
mgr	Command will return valid response (non RC16) when requested from network manager
any	Command will return valid response (non RC16) when requested via mote serial API, network manager, or from the HART Gateway session.

Legend for IA-510 Mote HART Command List

CMD	Description	Termination	Access	Comments
768	Write Join Key	mt	any	
769	Read Join Status	mt	any	
770	Request Active Advertise	mt	any	
771	Force Join	mt	any	
772	Read Join Mode Configuration	mt	any	
773	Write Network ID	mt	mgr, ser	
774	Read Network ID	mt	any	
775	Write Network Tag			Not implemented (RC64)

776	Read Network Tag			Not implemented (RC64)
777	Read Wireless Device Capabilities	mt	any	
778	Read Battery Life	mt	any	
779	Report Device Health	mt	any	
780	Report Neighbor Health List	mt	any	
781	Read Device Nickname Address	mt	any	
782	Read Session List	mt	any	
783	Read Superframe List	mt	any	
784	Read Link List	mt	any	
785	Read Graph List	mt	any	
786	Read Neighbor Property Flag			
787	Report Neighbor Signal Levels	mt	any	
788	Alarm "Path Down”	mgr	n/a	Access Restricted (RC16)
789	Alarm "Source Route Failed”	mgr	n/a	Access Restricted (RC16)
790	Alarm "Graph Route Failed”	mgr	n/a	Access Restricted (RC16)
791	Alarm "Transport Layer Failed”	mgr	n/a	
793	Write UTC Time Mapping	mgr	mgr	
794	Read UTC Time Mapping	mt	any	
795	Write Timer Interval	mt	mgr	
796	Read Timer Interval	mt	any	
797	Write Radio Power Output	mt	any	
798	Read Radio Output Power	mt	any	
799	Request Service	mgr	n/a	Access Restricted (RC16)
800	Read Service List	mt	any	
801	Delete Service	mgr, mt	mgr	May be sent mote->mgr or mgr->mote
802	Read Route List	mt	any	
803	Read Source-Route	mt	n/a	

804	Read CCA Mode	mt	any	
805	Write CCA Mode	mt	mgr	Access Restricted (RC16) for non-network manager
806	Read Handheld Superframe	mt	any	No Handheld Superframe (RC9)
807	Request Handheld Superframe Mode	mt		Not implemented (RC64)
808	Read Packet Time-To-Live	mt	any	
809	Write Packet Time-To-Live		mgr	Access Restricted (RC16) for non-network manager
810	Read Join Priority	mt	any	
811	Write Join Priority	mt	mgr	
812	Read Packet Receive Priority	mt	any	
813	Write Packet Receive Priority	mt	mgr	
814	Read Device List Entries			
815	Add Device List Table Entry			
816	Delete Device List Table Entry			Not implemented (RC64)
817	Read Channel Blacklist			Not implemented (RC64)
818	Write Channel Blacklist			
819	Read Back-Off Exponent	mt	any	
820	Write Back-Off Exponent	mt	mgr	
821	Write Network Access Mode			Not implemented (RC64)
822	Read Network Access Mode			Not implemented (RC64)
823	Request Session	mt		Not implemented (RC64)
832	Read Network Device Identity using Unique ID			Not implemented (RC64)
833	Read Network Device's Neighbor Health			Not implemented (RC64)
834	Read Network Topology Information			Not implemented (RC64)
835	Read Burst Message List			Not implemented (RC64)
836	Flush Cached Responses for a Device			Not implemented (RC64)

837	Write Update Notification Bit Mask for a Device			Not implemented (RC64)
838	Read Update Notification Bit Mask for a Device			Not implemented (RC64)
839	Change Notification			Not implemented (RC64)
840	Read Network Device's Statistics			Not implemented (RC64)
841	Read Network Device Identity using Nickname			Not implemented (RC64)
842	Write Network Device's Scheduling Flags			Not implemented (RC64)
843	Read Network Device's Scheduling Flags			Not implemented (RC64)
844	Read Network Constraints			Not implemented (RC64)
845	Write Network Constraints			Not implemented (RC64)
960	Disconnect Device	mt	mgr	
961	Write Network Key	mt	mgr	
962	Write Device Nickname Address	mt	mgr	
963	Write Session	mt	mgr	
964	Delete Session	mt	mgr	
965	Write Superframe	mt	mgr	
966	Delete Superframe	mt	mgr	
967	Write Link	mt	mgr	
968	Delete Link	mt	mgr	
969	Write Graph Edge	mt	mgr	
970	Delete Graph Connection	mt	mgr	
971	Write Neighbor Property Flag	mt	mgr	
972	Suspend Device(s)			Not implemented (RC64)
973	Write Service	mt	mgr	
974	Write Route	mt	mgr	
975	Delete Route	mt	mgr	

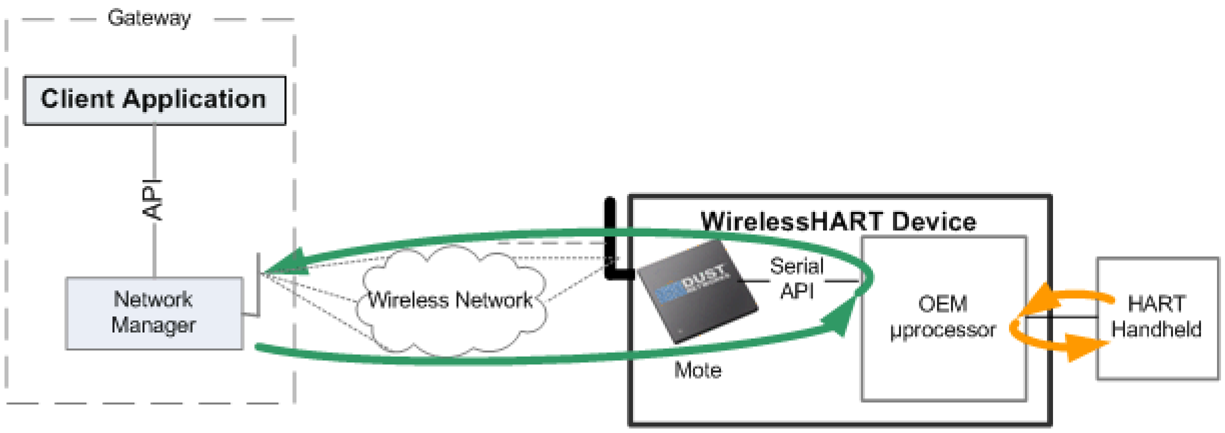
976	Write Source-Route			Not implemented (RC64)
977	Delete Source Route			Not implemented (RC64)
64,512	Read Wireless Module Revision	mt	any	

Mote WirelessHART Command List

Microprocessor-terminated Commands

The OEM microprocessor is responsible for terminating the non-wireless HART commands, which are not contained in HCF_SPEC-155. Of the ones the OEM microprocessor terminates, there are few commands for which the OEM microprocessor will need to communicate with the mote to properly implement the command.

The OEM microprocessor can receive and reply to HART commands over the air via the mote serial API (using the *dataReceived* and *send* mote serial API commands) or in most cases, through a serial port (for example, via a HART Handheld).



Sources of Microprocessor-terminated HART Commands
When a mote receives from the network a HART command that it does not terminate, it forwards the HART command to microprocessor via the *dataReceived* command.

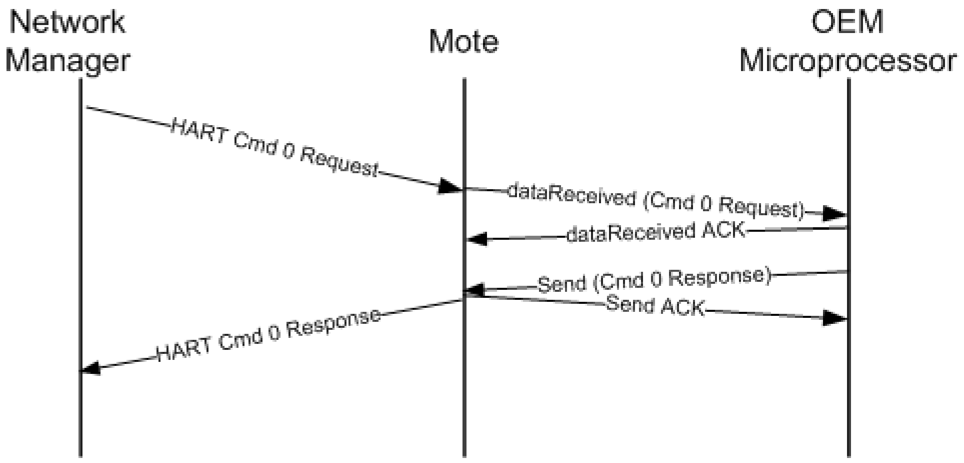
API support

dataReceived Notification, send Command

When the mote receives a HART packet over the air, it parses the command and determines whether it terminates the command (see the table in Command Termination). If the mote does not terminate the HART command, it passes the command to the OEM microprocessor via the *dataReceived* command.

The microprocessor must respond using the *send* command. Note that the mote will extract copies of the device status and extended device status from the HART payload, and use those values in its HART packets. Refer to the SmartMesh WirelessHART Mote Serial API Guide for more information about the *dataReceived* command and details on handling reliable downstream retries.

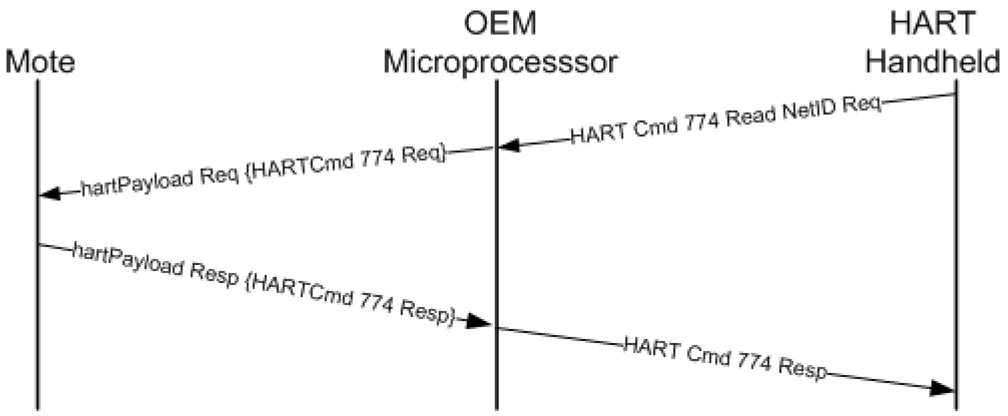
The following diagram shows a typical downstream request-response transaction.



Transaction Diagram–Typical Request/Response

hartPayload Command

The *hartPayload* mote serial API command allows the microprocessor to forward a HART command to the mote over the serial connection. These HART requests can come from the microprocessor, or they can be forwarded from a device that is external to the field device (for example, a HART Handheld). The preceding section describes how HART commands are handled via the *hartPayload* command. Refer to the SmartMesh WirelessHART Mote Serial API Guide for more information regarding command syntax and behavior.


Mote
OEM Microprocesssor
HART Handheld
HART Cmd 774 Read NetID Req
hartPayload Req {HARTCmd 774 Req}
hartPayload Resp {HARTCmd 774 Resp}
HART Cmd 774 Resp

Transaction Diagram–Using hartPayload Command for HART Handhelds

Typical Error Handling

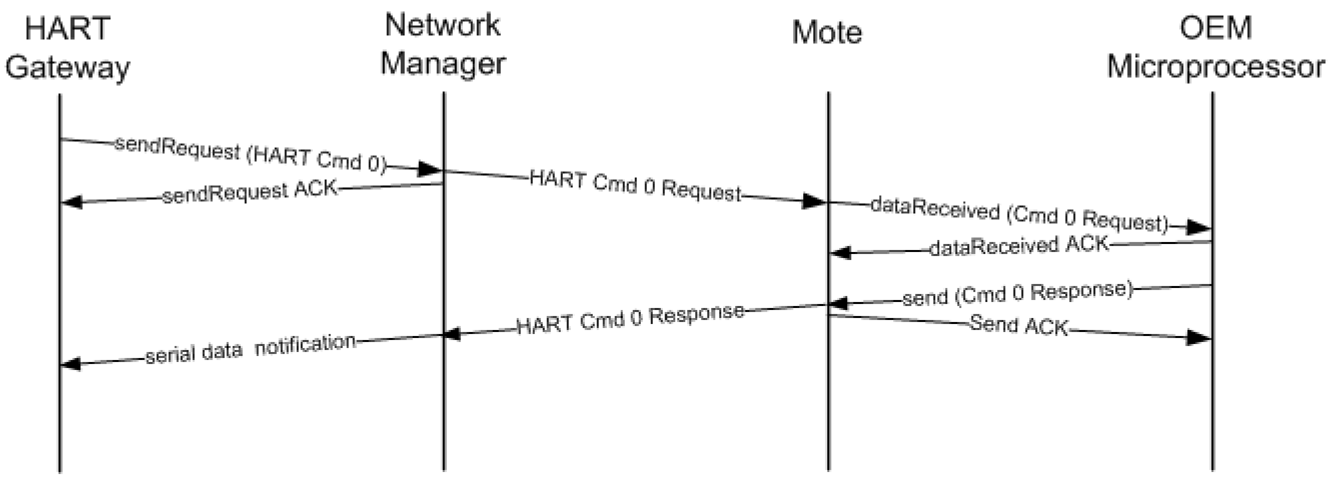
As explained in the SmartMesh WirelessHART Mote Serial API Guide, for every reliable request received by the *dataReceived* command, the OEM microprocessor must respond.

- If the OEM microprocessor receives a well-formed HART command that it does not terminate, it should reply with a reliable response packet via the send command with a HART packet containing HART response code 64, “Not Implemented.”
- If the OEM microprocessor receives a packet that is not a well-formed HART command, it should reply with a reliable response packet via the *send* command with empty payload.

Example of an End-to-end Transaction

The following is an example of an end-to-end downstream request-response transaction. In this example, the gateway uses the serial API *sendRequest* command to send HART Command 0 to the network manager. The network manager acknowledges receipt of the *sendRequest* and sends the HART packet over-the-air to the mote. After determining that it does not terminate this HART command, the mote uses the *dataReceived* command to send the HART command request to the OEM microprocessor. The OEM microprocessor acknowledges receipt of the *dataReceived* command.

To respond to the HART command, the OEM microprocessor uses the send command to send its response to the mote. The mote determines that it does not terminate this HART command and sends the command on to the network manager, which then sends the gateway a serial data notification containing the command response.



End-to-end Transaction Diagram

5.8.2 Key WirelessHART Command Support

HART Join Shed Time

The WirelessHART specification defines join shed time, which governs how the mote searches for neighbors during the join process. Dust Networks has engineered a more power-efficient means of searching for neighbors that is controlled by the *setParameter<joinDutyCycle>* command and does not affect HART device interoperability. It is recommended to use the default value for joinDutyCycle to reach the optimum join performance for mote power consumption. However, if the OEM wishes to implement join shed time as defined in the HART spec, the following steps illustrate how to use the mote serial API to accomplish this task:

1. The OEM microprocessor should read the join shed time from the mote using *hartPayload* (CMD 772 Read Join Mode).
2. While the mote is in the **Idle** state, issue the *setParameter<joinDutyCycle>* command, setting duty cycle to 100% (listen for neighbors 100% of the time).
3. Issue the *join* command.
4. After the OEM microprocessor has determined that join shed time has elapsed, issue the *setParameter<joinDutyCycle>* command to set a lower percentage of OEM's choosing.

Wireless Operation Mode and Join Process Status

The following table shows how the mote states map to the WirelessHART Operation Mode states and the Join status states as defined in HCF_SPEC-183 Common Tables Specification, Table 51, Wireless Operation Mode and Table 52, Join Process Status.

Hart Operation Mode Code	HART Operation Mode State	Dust State	HART Join Status	HART Join Status Description	Dust Join Status Description
0	Idle	Idle			
1 or 6	Active Search (1): duty cycle=100%. Passive search (6): duty cycle<100%.	Searching	0x001	Network Packets Heard.	Set when first packet with matching Network ID is heard.
			0x002	ASN Acquired.	Set when first advertisement is processed.
			0x004	Synchronized to Slot Time.	Set when first advertisement is processed.
			0x008	Advertisement Heard.	Set when first advertisement is processed.
2	Negotiating	Negotiating	0x010	Join Requested. Set on transmission of first join request.	Same as HART description.

			0x020	Join Retrying. Set after the first join request retry (when number of join requests is greater than two). Cleared when device is Authenticated or when Active Search is started.	Same as HART description.
			0x040	Join Failed. Set on transition from Active Search mode to Passive Search. Cleared on transition from Passive to Active Search. Mote will move to HART Operational mode 6 - "Passive Search." Note: HART uses the term "Deep Sleep" in Spec 183 table 52 instead of Passive Search.	Set when mote stops join attempts.
3	Quarantine	Connected	0x080	Authenticated. Network Key, Network Manager Session Established.	Same as HART description.
			0x100	Network Joined. Normal superframe and links obtained.	Set when mote stops using its join links.
4	Operational	Operational	0x200	Negotiating Network Properties. Gateway session obtained. Initial Bandwidth requirements being negotiated with Network Manager.	Set when the gateway session is created.
			0x400	Normal Operation Commencing.	Set when the gateway session is created.

Wireless Operation Mode and Join Process Status

Device Status Support

HCF_SPEC-085 specifies the transport layer header attached to all HART packets. Two key parts of the transport layer header, the **device status** and the **extended device status** fields, pertain to field device status and are therefore owned by the OEM microprocessor. The OEM microprocessor must periodically send the mote the device status and extended device status. To streamline this process, the mote serial API supports two ways of sending the mote the status values:

- *setParameter<hartDeviceStatus>* command
- *send* command (mote parses HART payload and caches status and extended status)

Refer to the SmartMesh WirelessHART Mote API Guide for command details.

CMD 770 Request Activate Advertising

CMD 770, Request Activate Advertising is unique in how it is defined in HCF_SPEC- 155. HART allows this command to be addressed either to the field device or to the network manager so that the same command can be forwarded from the field device to network manager. The mote terminates CMD 770, Request Activate Advertising, and in turn will send a request to the network manager (in the form of a CMD 770 request) to activate advertising. If the OEM microprocessor receives a CMD 770 (for example, from a HART handheld), it should forward CMD 770 to the mote using the *hartPayload* mote serial API command. It should not use the *send* mote serial API command to send it directly to the network manager, even if the handheld addresses the original packet to the network manager.

When the mote receives CMD 770, it responds with the HART response code 33, “DR Initiated,” and forwards the CMD 770 request to the manager. When the mote receives a response from the manager, it cashes the response values. The OEM microprocessor should periodically send CMD 770 to the mote until the delayed response has completed. If mote is still waiting for the manager’s response, it responds with HART response code 34, “DR Running.” Similarly, if the manager responds with HART code 33, “DR Initiated,” or code 34, “DR Running,” the mote will respond with the code 34 and forward another request to the manager. This process continues until the delayed response is completed, resulting in the mote sending the OEM microprocessor a HART response code other DR Initiated or DR Running. If the final result is not read from the mote within 10 minutes after the delayed response was initiated, the delayed response times out and the next CMD 770 will be treated as a new request rather than returning the previous cached result.

The HART response code 32, “DR Busy,” code 35 “DR Dead,” and code 36, “DR Conflict” have the following meaning for the mote:

- DR Busy—This code indicates that mote does not have sufficient resources to forward CMD 770 to the manager.
- DR Dead—If the manager responds using DR Busy or DR Conflict, mote will translate the response code to DR Dead.
- DR Conflict—If a delayed response is initiated and the next CMD 770 request contains a modified shed time and is received while a request is outstanding to the manager, the mote responds with the code DR Conflict. If no manager request is outstanding, mote will treat this request as new and respond with DR Initiated.

CMD 771 Force Join Mode

CMD 771 Force Join Mode sets the join mode parameter, which controls the joining behavior of the mote as defined in the HCF_SPEC-183 Common Tables Specification, Table 61, Join Mode Code. The mote stores join mode in its non-volatile memory and by Dust factory default, the join mode is equal to 2, enabling it to join. While this is typically used by network manager, the OEM microprocessor can use CMD 771 to:

- Force mote to rejoin-CMD 771 with join mode equal to 1
- Disallow mote to join-CMD 771 with join mode equal to 0
- Enable a mote to join-CMD 771 with join mode equal to 2

When the mote receives command 771, it will update its local values (in non-volatile memory) for both join mode and shed time. These values will be used the next time the mote receives the *join* API command from the **Idle** state. In the case where the join mode is equal to 1 (join now), the mote will store join mode value 2 (per the spec), reboot and go through the normal boot sequence.

Using the mote serial API commands *disconnect* then *join* will result in the mote rebooting and attempting to initiate join. The mote will still obey whatever join mode value that has been previously set. For example, if the mote is in join mode 0, then the mote will not join when it receives the *join* API command. In contrast, a CMD 771 with join mode equal to 1 will both update the join mode value and force a mote to attempt to join by putting the mote in the **Idle** state and having it follow the normal join sequence. See the diagram in Connecting and Disconnecting.

CMD 797 Write Radio Power Output and CMD 798 Read Radio Power Output

The mote terminates CMD 797 Write Radio Power Output and CMD 798 Read Radio Power Output. These commands specify the Equivalent Isotropic Radiated Power (EIRP) RF output energy of the entire field device, where: EIRP = conducted output power + (antenna Gain - attenuation from connection) For example, if a mote has an 8 dBm conducted output connected to a 2 dBi antenna via a low loss coaxial cable, this will yield 10 dBm output EIRP. The mote serial API includes the command *setNVParameter<HARTantennaGain>* that enables the microprocessor to input the "(antenna Gain - attenuation from connection)" to the mote.

Refer to the SmartMesh WirelessHART Mote API Guide for syntax and default values.

The RF output power can be controlled by an OEM microprocessor via the hartPayload<CMD 797>, which is specified in EIRP, or via the native mote serial API commands, *setParameter<txPower>*, *setNVParameter<txPower>*, and *getNVParameter<txPower>*, which are in conducted RF output power

Write Protect

The HART specification allows a field device to be in write protect mode. When in this mode, certain HART write commands must be disallowed. The mote serial API includes a command *setParameter<writeProtect>* to enable and disable write protection of mote-related values. The following table lists the commands that are write protected when write protect mode is enabled.

HART CMD	Description
768	Write join key
770	Request active advertising
773	Write Network ID
797	Write radio power output
805	Write CCA mode
809	Write packet time-to-live

Change Counter and Change Configuration Flag

The mote maintains an internal *changeCounter* variable that it increments each time specific HCF_SPEC-155 commands is successfully executed. For a list of commands that trigger the *changeCounter*, refer to the Wireless Command Specification (HCF_SPEC-155).

The *changeCounter* counter is 16 bits. It is initialized to 0 when the mote reboots and rolls over back to 0 after reaching 0xFFFF. The counter can be retrieved via the *getParameter<moteStatus>* command. Additionally, the mote maintains an internal change configuration flag (CCF) that it sets whenever it increments the *changeCounter* variable. When the bit is set, a *configChanged* event is generated. The mote's CCF is ORed into hart status bytes sent with every packet. In particular, the CCF will be set in response to the request that caused the flag to be set.

The mote clears the CCF bit when the *configChanged* event is acknowledged. To implement the HART change counter and the CCF, the OEM microprocessor should add its own change counter to the mote's change counter for reporting in command 0. It should also set the CCF whenever it receives a *configChanged* event from the mote.

CMD 42 Device Reset

When the OEM device receives CMD 42 Reset Device, it should use the disconnect command (not the reset command) to request that the mote leave the network. With the *disconnect* command the mote informs its neighbors that it is about to become unavailable before it resets. The *reset* command simply resets the mote without notice. For more information on disconnecting from the network, see Connecting and Disconnecting.

5.8.3 Efficiently Checking for Manager-Originated Services

The OEM microprocessor can efficiently stay up to date on manager-originated services because the mote proactively sends a *serviceIndication* packet when services are added, deleted, or modified. However, the OEM microprocessor always has the option of polling the service table to verify service state. Manager-originated services are defined by HART to have service ID between 0x80 and 0xFF. Although SmartMesh WirelessHART network managers will specifically use ID 0x80 for maintenance service, network managers from other vendors may not allocate manager-originated services in a read-friendly order.

To efficiently check the service table for manager-originated services:

1. Determine the number of active manager requested services.
    1. Read the number of services in use—issue an API command *hartPayload* with HART command 800.
    2. Subtract the number of device-originated services (number of services the OEM microprocessor has requested, which will have service IDs between 0x00 and 0x7F).
2. Read through the service table, starting with 0x80 (128) and iterate upwards through the service IDs until either the number of active manager services has been found, or the end of the table (ID 0xFF) has been reached.

The process described above works very efficiently with SmartMesh WirelessHART network managers. Again, the above solution for iterating through the manager-originated portion of the service table is not required, but is offered in the event an OEM should choose to use it.

5.9 Factory Default Settings

The mote ships with the following factory default settings. The mote can be returned to factory settings by using the `mclearnv` mote CLI command, or the *clearNV* mote API command.

Parameter	Default Value
MAC Address	Use 8-byte UID value from OTP
Transmit Power	+8 dBm
Network ID	1229
Join Key	0x445553544E4554574F524B53524F434B
Nonce Counter	0
Join Duty Cycle	5%
Power Source	Battery
Discharge Current	1000 µA (1 mA)
Discharge Time	42949667295 1/32 ms (0xFFFFFFFF)
Recover Time	0
TTL	127
HART Antenna Gain	+2 dBi
OTAP Lockout	0

5.10 Master vs. Slave

5.10.1 Modes

Motes have two modes that control joining and command termination behavior:

- **Master** - a demo mode enabled on the motes in Starter kits. In this mode, the mote runs an application that generates sample data and controls joining. The mote API is disabled in **master** mode.
- **Slave** - the default mode for LTC58xx and LTP59xx motes. The mote expects a serially connected device to terminate commands and control join - by default the mote does not join a network on its own. The API is enabled in **slave** mode, and the device expects a serially attached application such as APIExplorer or an external microcontroller to connect to it.

The mode can be set through the CLI `set` command, and persists through reset (*i.e.* it is non-volatile).

If *autojoin* is enabled via *SetParameter* (SmartMesh IP only), a **slave** mote will join the network without requiring a serial application to issue a *join* command in order to simplify external microcontroller logic. Do not use the *autojoin* parameter with a mote in **master** mode, as it may become unresponsive in some revisions of software.

5.10.2 LEDs

For motes (DC9003) in **master** mode, the STATUS_0 LED will begin blinking immediately upon power-up, as the mote will start searching automatically. When the mote has joined, STATUS_0 and STATUS_1 LEDs will both be illuminated. In **slave** mode, no LEDs light - this should not be mistaken for a dead battery.

LEDs of a DC9003 board will only light if the LED_EN jumper is shorted. Master mode LED support available in SmartMesh WirelessHART mote version >= 1.1.2.

5.10.3 Master Behavior

The **master** mode application is not WirelessHART compliant:

- It does not follow the Pre-join configuration requirements
- It generates temperature reports that do not contain status or extended status bytes

Only temperature and packet generation are available in the WirelessHART Master mode application.By default, the application will sample and send temperature reports every 30 s. An example of the format of the data notification payload field is as follows:

Non-HART Data	OAP Header	Notification type	Channel	Timestamp	Rate	# samples	Sample size	Sample(s)
FC12	0X YY 05	00	FF 01 05	00 00 00 00 53 16 60 93 00 04 e5 77	00 00 75 30	01	10	0ae0

OAP Header

- Control: 00 (unacknowledged request, normal sync) or 02 (unacknowledged request, resync)
- ID: YY (ID can be ignored for unacknowledged data)
- Command: 05 (notification)

OAP Payload (Notification)

- Type: 00 (raw samples)
- Channel: FF 01 05 (tag, length, value=5, i.e. temperature)
- UTC Timestamp: 00 00 00 00 53 16 60 93, 00 04 e5 77 (seconds into epoch, microseconds)
- Rate: 00 00 75 30 (30000 milliseconds)
- Number of samples: 01
- Sample size: 10 (16 bits)
- Samples: 0a e0 (2784 100ths of a °C)

Changing Reporting Rate / Enabling Packet Generator

Both temperature and packet generation can be enabled/disabled, and the rate set by encapsulating OAP messages in the payload (prepended with 00 00 FC 12) of a Manager *sendRequest* API. See the SmartMesh IP Tools Guide for details on OAP.

5.10.4 Switching To Slave Mode

By default, motes in starter kits (DC9000 & DC9021 and DC9007) and are configured for **master** mode. To read the current configuration, connect the mote to a computer via a USB cable and use the `get` mote CLI command. To configure the mote for **slave** mode, use the `set` mote CLI command:

Use the `get mode` command to see the current mode:

```
> get mode
master
```

Use the `set mode` command to switch to **slave** mode:

```
> set mode slave
> reset
```

⚠ You must reset the mote for the mode change to take effect. Once set, the mode persists through reset.

5.10.5 Switching To Master Mode

To read the current configuration, connect the mote to a computer via a USB cable and use the `get mode` CLI command. To configure the mote for **master** mode, use the `set mode` CLI command.

Use the `get mode` command to see the current mode:

```
> get mode
slave
```

Use the `set mode` command to set the mote to **master** mode**:**

```
> set mode master
> reset
```

⚠ You must reset the mote for the `set mode` command to take effect. Once set, the mode persists through reset.