

Dust Networks

SmartMesh[®] IA-510 Manager Serial API Guide

WirelessHART[®]



Trademarks

SmartMesh-XR, SmartMesh-XT, SmartMesh-XD, and SmartMesh IA-510 are trademarks of Dust Networks, Inc. The Dust Networks logo, Dust, Dust Networks, and SmartMesh are registered trademarks of Dust Networks, Inc. All third-party brand and product names are the trademarks of their respective owners and are used solely for informational purposes.

Copyright

This documentation is protected by United States and international copyright and other intellectual and industrial property laws. It is solely owned by Dust Networks, Inc. and its licensors and is distributed under a restrictive license. This product, or any portion thereof, may not be used, copied, modified, reverse assembled, reverse compiled, reverse engineered, distributed, or redistributed in any form by any means without the prior written authorization of Dust Networks, Inc.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g) (2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015 (b)(6/95) and DFAR 227.7202-3(a), and any and all similar and successor legislation and regulation.

Disclaimer

This documentation is provided “as is” without warranty of any kind, either expressed or implied, including but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

This documentation might include technical inaccuracies or other errors. Corrections and improvements might be incorporated in new versions of the documentation.

Dust Networks does not assume any liability arising out of the application or use of any products or services and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

Dust Networks products are not designed for use in life support appliances, devices, or other systems where malfunction can reasonably be expected to result in significant personal injury to the user, or as a critical component in any life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Dust Networks customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify and hold Dust Networks and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Dust Networks was negligent regarding the design or manufacture of its products.

Dust Networks reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products or services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to Dust Network's terms and conditions of sale supplied at the time of order acknowledgment or sale.

Dust Networks does not warrant or represent that any license, either express or implied, is granted under any Dust Networks patent right, copyright, mask work right, or other Dust Networks intellectual property right relating to any combination, machine, or process in which Dust Networks products or services are used. Information published by Dust Networks regarding third-party products or services does not constitute a license from Dust Networks to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from Dust Networks under the patents or other intellectual property of Dust Networks.

© Dust Networks, Inc. 2007, 2008, 2009, 2010. All Rights Reserved

Document Number: 040-0046 rev 10 SmartMesh IA-510 (H) Manager Serial API Guide

Last Revised: October 25, 2010

Document Status	Product Status	Definition
Advanced Information	Planned or under development	This document contains the design specifications for product development. Dust Networks reserves the right to change specifications in any manner without notice.
Preliminary	Engineering samples and pre-production prototypes	This document contains preliminary data; supplementary data will be published at a later time. Dust Networks reserves the right to make changes at any time without notice in order to improve design and supply the best possible product. The product is not fully qualified at this point.
No Identification Noted	Full production	This document contains the final specifications. Dust Networks reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
Obsolete	Not in production	This document contains specifications for a product that has been discontinued by Dust Networks. The document is printed for reference information only.

Contents

About This Guide

Related Documents	iii
Conventions Used	iii
Revision History	iv

1 Introduction

SmartMesh-enabled Networks	1
Serial API	2
Serial Port Process Switch	2

2 Using the Serial API

Packet Communication and Format	5
Data Types	6
Byte Ordering	7
Packet Field Descriptions	7
Control	7
Type	8
Sequence Number	8
Payload	8
Establishing a Session	8
Establishing a Serial API Session	8
Establishing a New Session after Client Reboot	9
Detecting a Manager Reset	10
Client State Machine	10
Manager State Machine	11
Session Establishment Commands	12
MGR_HELLO	12
HELLO	13
HELLO_RESPONSE	13
Session Packet Processing	14
Best-effort Communication (non-ACK)	14

Best-effort Packets	15
Reliable Communication	16
Reliable Packets	16
ACK Packets	17
Full-duplex Communication	17
Processing Guidelines	18
Example of Failed Communication	18
Error Handling	19
End-to-end Wireless Communication	19
Reliable Wireless Communication	19
Best-effort Wireless Communication	19
Network Bandwidth Control	20
Network ID	22
Network Security	23
Common Join Key Mode	23
ACL Mode	23
Field Mask	24
Mote Joining	25
Response Codes	25
Commands	27
activateAdvertising	30
activateFastPipe	31
cancelOtap	32
cli	32
deactivateFastPipe	33
decommissionDevice	34
deleteAcIDevice	35
deleteMote	35
disconnect	36
exchangeJoinKey	37
exchangeMoteJoinKey	38
exchangeNetworkId	39
exchangeNetworkKey	41
exchangeSessionKey	42
getAcIDevice	43
getChannelBlacklist	44
getLatency	45
getMote	46
Mote States	49
getMoteStatistics	49
getNetStatistics	51

getNetwork	52
Bandwidth Profiles	55
getNextAcIDevice	55
getNextAlarm	56
Alarm Types	57
getNextMote	57
getNextOtapMote	59
getNextPath	60
getOtapFile	60
getOtapMote	61
Mote OTAP State	62
getOtapStatus	63
getPath	64
getPathStatistics	65
getPrevEvent	66
getSecurity	67
getSLA	69
getSystem	69
getTime	71
pingMote	72
reset	72
sendRequest	74
sendResponse	75
setAcIDevice	76
setChannelBlacklist	77
setMote	78
setNetwork	81
setSecurity	84
setSLA	85
setSystem	86
startOtap	88
subscribe	88
Notifications	90
Notification Packet Format	91
Notification Details	91
Serial Data Notification Packet	92
Alarm Notification Packet	92
System Event Notification Packet	95
Network Event Notification Packet	96
System Error Notification Packet	96
Log Notification Packet	96
CLI Notification Packet	97
Events	97
System Events	98

Network Events	102
A Serial Port Login	
Manager Serial API Login	109
PPP Login	109
Linux Shell Login	111
Manager CLI Login	112
B HDLC Packet Format	
HDLC Packet	113
Start/Stop Delimiters	113
Information	113
Frame Check Sequence	113
Octet Stuffing	113
HDLC Packet Processing Examples	114
C Configuring a Windows XP PC for PPP	
D Updating Software	
Transferring a Software Update Package to the Manager	121
Initiating a Mote Software Update Package	122
Initiating a Manager Software Update	124
E Adjusting Manager Time	
adjustTime	125
Index	


About This Guide

This guide provides instructions for using the SmartMesh manager (SMM) serial protocol to communicate with the manager over its asynchronous serial port.

Related Documents

The following documents are available for SmartMesh-enabled networks:

- *SmartMesh Manager Serial API Guide* (this guide)
- *SmartMesh Manager XML API Guide*


 **Note:** For information on configuring and connecting to the serial API, refer to the product datasheet for embedded managers (PMxxxx) or user guide for packaged managers (Dxxxx).


Conventions Used

The following conventions are used in this guide:

- `Computer type` indicates information that you enter, such as specifying a URL.
- **Bold type** indicates buttons, fields, and menu commands.
- *Italic type* is used to indicate a command, notification, or field, and to introduce a new term.

 **Note:** Notes provide more detailed information about concepts or consequences.

 **Caution:** Cautions advise you about actions that might result in a loss of data.

 **Warning!** Warnings advise you about actions that may cause physical harm to the hardware or your person.

Revision History

Revision	Date	Description
040-0046 rev 1	10/1/2007	Advanced information (pre-product release).
040-0046 rev 2	12/14/2007	Preliminary information (pre-product release).
040-0046 rev 3	6/16/2008	Preliminary information (pre-product release).
040-0046 rev 4	6/25/2008	Final publication (product release).
040-0046 rev 5	6/27/2008	
040-0046 rev 6	8/8/2008	
040-0046 rev 7	10/3/2008	
040-0046 rev 8	10/5/2009	
040-0046 rev 9	11/6/2009	
040-0046 rev 10	10/25/2010	

Introduction

This chapter introduces the SmartMesh manager (SMM) serial application programming interface (API) and describes the SmartMesh-enabled network.

SmartMesh-enabled Networks

A SmartMesh-enabled network contains:

- A network of motes (remote sensors) that sense data and exchange information packets via radio.
- A manager that configures the network and motes, receives data packets from the motes, and streams data to a client application.
- The OEM controller that provides the manager with configuration instructions and receives mote and network data from the manager in serial format.

The OEM controller exchanges serial packets with the manager, which in turn communicates wirelessly with the motes in the network.

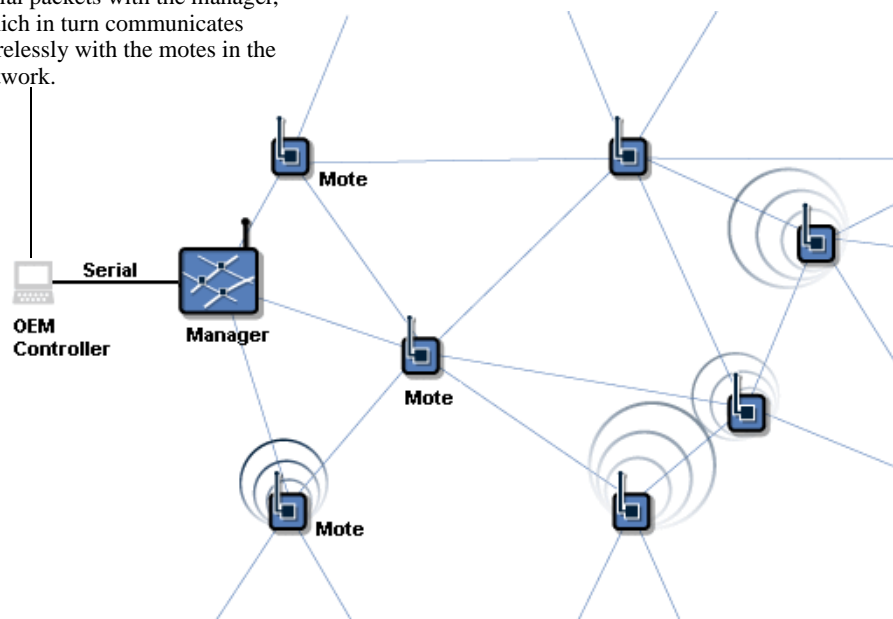


Figure 1 A SmartMesh-enabled Network

Serial API

The SmartMesh manager serial API is a packet-based serial protocol that allows client applications to communicate with the manager over its asynchronous serial port. It is used for managing and configuring a SmartMesh wireless network and for sending serial data to and receiving data from wireless motes. The serial API covers the same features available through the TCP/IP-based XML-RPC interface.

The serial API provides two types of serial communication:

- **Commands**—are two-way command request and response messages. Commands are used to send serial data to wireless motes, configure and manage the wireless network, and subscribe to the notification channel.
- **Notifications**—are asynchronous messages sent from the manager to the client. Notifications are used to receive serial data from wireless motes and receive network events and alarms.

For a client use the serial API to communicate with the manager, the client must first establish a serial session with the manager. Establishing a serial session is described in Chapter 2, along with detailed information about packet format, packet processing, data types, commands, and notifications.

Serial Port Process Switch

The serial port runs a terminal connection program that serves as a process switch for the controller, allowing the controller to select the functionality of the serial port at runtime. The controller can initiate a PPP, manager serial API, manager CLI, or Linux shell process over the serial port. The login user name and password determines which process is initiated.

The process switch allows you to perform a variety of operations via the serial port, including:

- Sending and receiving wireless data
- Configuring and administering the network
- Transferring files to and from the manager
- Initiating software updates

The controller can detect the terminal connection by recognizing the ascii character string “manager login:”. As indicated in Figure 2, the session establishment procedure is different for each process. Detailed login instructions are provided in Appendix A. For additional information about the serial port, including the default data rate, refer to the *SmartMesh IA-510 PM2511 Datasheet*.

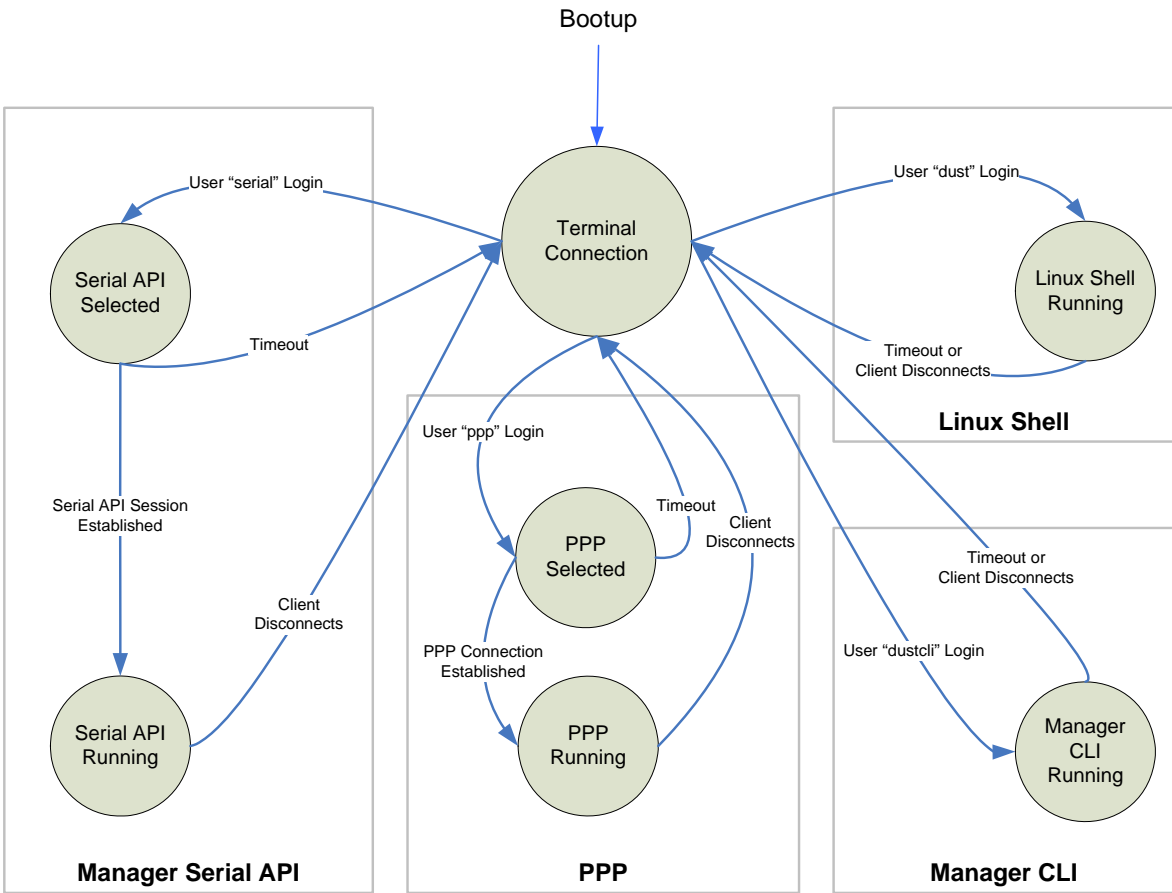


Figure 2 Serial Port Login Processes

Using the Serial API

This chapter describes the packet format, packet processing, serial commands, and serial notifications. It also provides instructions for establishing a session between the client and the manager.

Packet Communication and Format

The IA-510 manager serial protocol uses byte-oriented HDLC framing for all packets, as outlined in RFC 1662 (see <http://rfc.net/rfc1662.html>). The following diagram summarizes the packet format. Appendix B provides a description of HDLC fields and examples of HDLC packet processing.

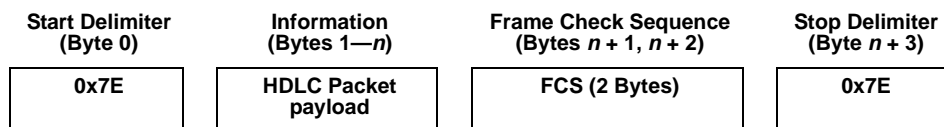


Figure 3 HDLC Packet Format

All packets in the IA-510 manager serial protocol contain the following fields in the Information section of HDLC packet.

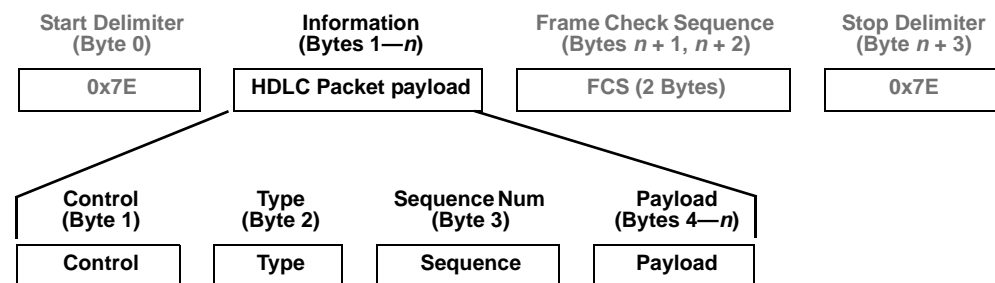


Figure 4 IA-510 Protocol Packet Format

Data Types

Packet formats described in this document make use of the following data types. These values are reflected in the Data Type column of the packet format description. Binary structures are represented by a packed binary format.

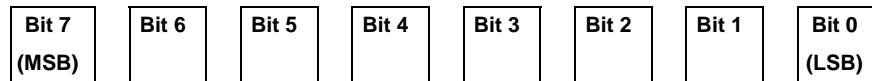
Table 1 Data Types

Data Type	Description
Byte	Single byte.
Byte[n]	Fixed size array. Fixed size arrays, such as MAC address (Byte[8]), always contain [n] elements. For fixed size arrays that may contain fewer valid values, a default value is specified.
Byte[]	Variable length array. The size of variable length arrays is determined by the length of the packet. Variable length arrays are always the last field in a packet structure.
Char	Single string character (1 byte).
Char[n]	Fixed length character array. Strings specified as fixed length character arrays (for example, Char[33]) must be terminated with at least one null character. Values shorter than the full size must be padded with null characters.
Char[]	String character array. Strings specified as character arrays are terminated with the null character.
ShortInt	Short integer (2 bytes).
LongInt	Long integer (4 bytes).
FixedPoint	Fixed point number (4 bytes), where bytes 1-2 (ShortInt) contain the integral part of the number, and bytes 3-4 (ShortInt) contain the fractional part of the number 65536. For example, the fractional part of the value 0.5 would be represented as 32768 ($65536 \times 0.5 = 32768$). For byte ordering, see "Fixed Point Number" on page 7.
DustTime	<p>The DustTime structure represents time as the number of microseconds since midnight January 1, 1970, separated into the seconds and microseconds component.</p> <div style="text-align: center;"> <pre> graph TD DT["DustTime (Bytes 1-8)"] --- SE["Seconds Since Epoch (Bytes 1-4) ULong"] DT --- MS["Microseconds (Bytes 5-8) ULong"] </pre> </div>
ASN	The ASN (absolute slot number) is the number of timeslots since network startup.

Byte Ordering

All values sent over the SMM serial protocol should be sent in big-endian order. The following are the representations of commonly used fields. The data appears on the wire MSB-first, as shown below:

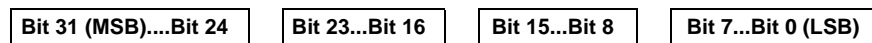
Byte



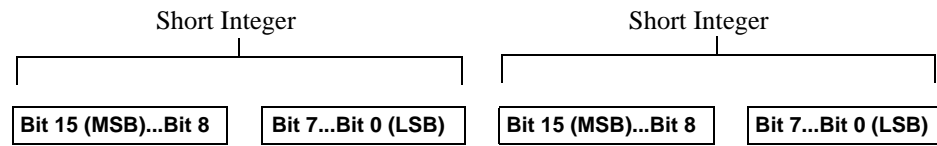
Short Integer



Long Integer



Fixed Point Number



Packet Field Descriptions

Control

Control is a mandatory bitmap field contained in the beginning of every packet. It describes whether the packet is a data packet or an acknowledgement packet, and whether to use best-effort or reliable serial communications. These terms are described in detail in “Session Packet Processing” on page 14.

The following bits are defined for the *control* field.

Field	Definition
Bits 2–7	Reserved, set to 0
Bit 1 (SERVICE_TYPE)	0 = best effort 1 = reliable
Bit 0 (DATA/ACK)	0 = data packet (non-ACK) 1 = ACK packet

Type

Type is a mandatory field indicating the type of packet that follows. Each command, notification, and session establishment packet has a unique type number. The exact packet format depends on this field's value, and is illustrated in the following sections for each command and notification.

Sequence Number

The sequence number (*sequence* field) is used only for reliable data communication. In regular (non-ACK) packets, it contains the sender's unique reliable packet number (see "Reliable Communication" on page 16 for details). In reliable ACK packets, this field contains the sequence number of packet being acknowledged. In best-effort packets, this field is unused and set to zero.

Payload

The *payload* is an optional field, the presence and contents of which are determined by the TYPE field. Payload can be present in both regular (non-ACK) and ACK packets.

Establishing a Session

This section describes how a client must establish and maintain a serial session with a manager. Before establishing a session, the client must first log into the serial port as "serial" which provides access to the manager serial API. See Appendix A for detailed login instructions.

The *HELLO* packet exchange starts a new session between the client and the manager. The handshake is used to clear previous settings, agree on protocol version, and establish sequence numbers for future reliable communication.

Establishing a Serial API Session

Prior to the *HELLO* packet exchange, both the client and the manager should drop any non-*HELLO* packets received. A client should initiate the exchange in the following situations:

- To establish a new session.
- To re-establish session in case of communication failure.
- Upon receiving *MGR_HELLO* packet (see Figure 6).

To establish a new session with the manager:

- ① Generate periodic *HELLO* packets (once per second is the recommended interval). For each new packet generated, increment the *clientSeqNum*.
- ② If a valid *HELLO_RESPONSE* packet is received (*successCode* is 0 and *clientSeqNum* matches the one in the *HELLO* packet just sent), the session is established and data can start flowing.

- ③ If an invalid `HELLO_RESPONSE` packet is received (`successCode` is not 0), the client can choose to adjust the `HELLO` message parameters (for example, `version` field) and resend.
- ④ If no `HELLO_RESPONSE` packet is received, the client should continue resending `HELLO` until reply is received.
- ⑤ Any other packet (including `MGR_HELLO`) received in response to `HELLO` packet must be dropped.

Upon completing `HELLO` exchange both sides should:

- Purge all packets pending for transmission.
- Signal the application layer to go into default state.
- Record own and other side's sequence numbers.

Establishing a New Session after Client Reboot

If a session is established between the client and the manager and the client reboots, the client should resend the `HELLO` packet to reestablish the session, as described in the steps in the preceding section. A client reboot will not cause the manager to send `MGR_HELLO` packets.

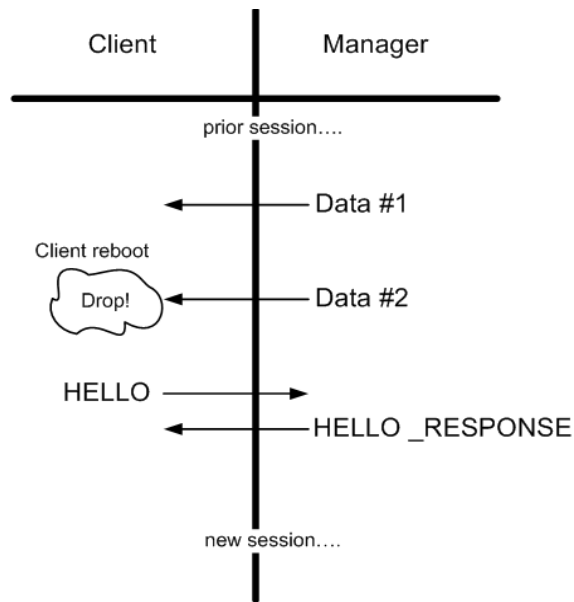


Figure 5 Establishing New Session Following Client Reboot

Detecting a Manager Reset

Whenever it is not participating in an established session, the manager sends periodic *MGR_HELLO* packets at the rate of once every three seconds. This state can occur either following the manager reset, or after the manager terminates its session due to communication errors.

Upon receiving a *MGR_HELLO* packet, the client should assume the previous session has been terminated, and attempt to reestablish connection using the *HELLO/HELLO_RESPONSE* exchange.

The following figure shows an example of how a session is reestablished following a manager reset.

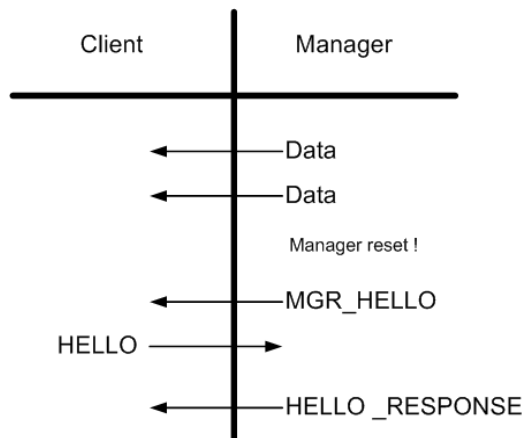


Figure 6 Hello Handshake Following a Manager Reset

Client State Machine

Figure 7 outlines the client's state machine for session management.

- **Unconnected**—Initial state after reset or following a communication failure. To initiate connection, the client sends *HELLO* message and moves into *HELLO_Sent* state.
 - ☞ **Note:** Communication failure is defined in “Error Handling” earlier in this chapter.
- **HELLO_Sent**—The client stays in this state while waiting for reply to its *HELLO* message. Receiving a valid *HELLO_RESPONSE* puts the client into Connected state. Timeout and errors cause the client to go back into Unconnected state.
- **Connected**—The client stays in this state during the session. Errors or detecting a manager reset (receiving *MGR_HELLO*) causes the client to go into Unconnected state.

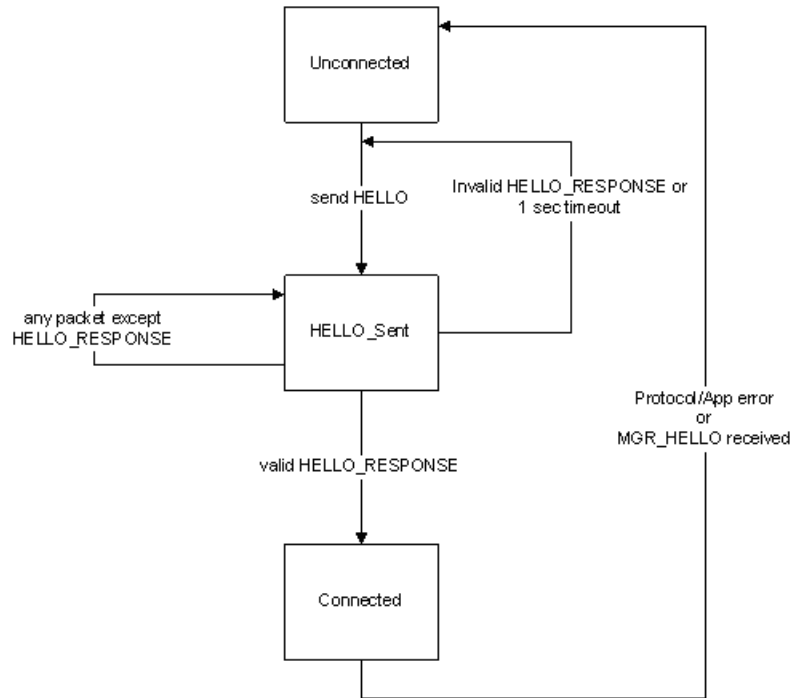


Figure 7 Client State Machine

Manager State Machine

Figure 8 outlines the manager session state machine.

- **Unconnected**—Initial state after reset or following communication failure. To notify the client of the manager reset, the manager sends *MGR_HELLO* packet and moves into *HELLO_Sent* state.

 **Note:** Communication failure is defined in “Error Handling” earlier in this chapter

- **HELLO_Sent**—The manager stays in this state following the transmission of its *MGR_HELLO* packet. Upon timeout, the packet is resent. Receiving a valid *HELLO* packet from the client causes the manager to send back *HELLO_RESPONSE* and move into the *Connected* state.
- **Connected**—The manager stays in this state during active session. Receiving a valid *HELLO* message from the client causes the manager to clear all of its settings/buffers and re-enter the *Connected* state. Receiving a protocol error causes the manager to move into the *Unconnected* state.

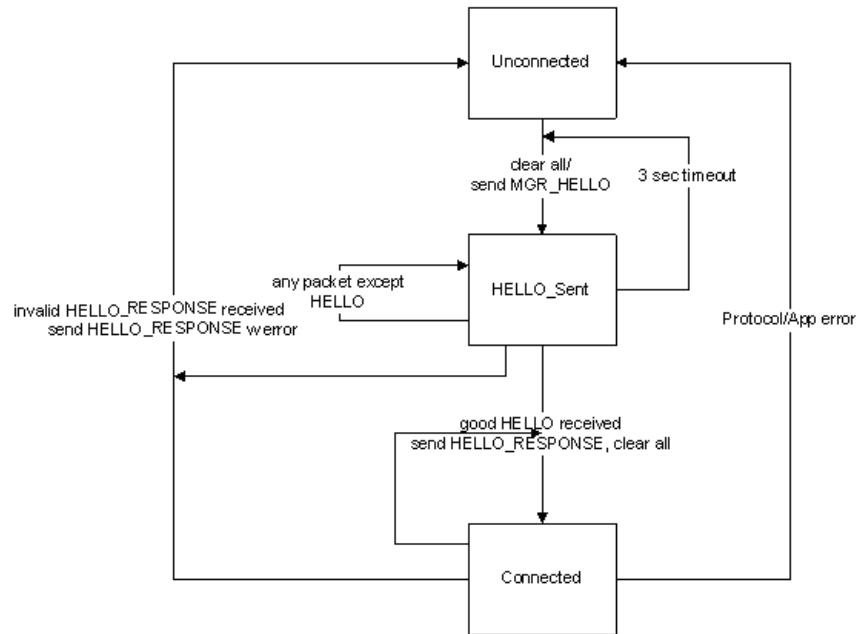


Figure 8 PM State Machine

Session Establishment Commands

This section describes the establishment commands. Table 2 indicates the *type* value for these commands.

Table 2 Session Establishment Commands

Type	Command	Description
0x01	HELLO	Establishes a serial session with the manager.
0x02	HELLO_RESPONSE	
0x03	MGR_HELLO	

MGR_HELLO

The *MGR_HELLO* is continually sent to the client until the client responds with the *HELLO* packet.

Table 3 MGR_HELLO

Message Byte	Description	Data Type	Value
1	Control	Byte	0x00
2	Type	Byte	0x03
3	Sequence	Byte	

HELLO

The *HELLO* packet is sent by the client to initiate new session with the manager.

Table 4 HELLO Packet

Message Byte	Description	Data Type	Value
1	Control	Byte	0x00
2	Type	Byte	0x01
3	Sequence	Byte	0x00
4	Version	Byte	
5	Client sequence num	Byte	

version—The version of IA-510 manager Serial Protocol supported by the client. The manager checks this field to decide on compatibility with the client. *client sequence number*—The unique number of this *HELLO* packet. Used for reliable communication only, both sides use unique numbers to detect duplicate reliable messages. The first reliable request packet from the client must carry the next sequence number. For details, see “HELLO_RESPONSE” below.

HELLO_RESPONSE

Table 5 HELLO_RESPONSE Packet

Message Byte	Description	Data Type	Value
1	Control	Byte	0x00
2	Type	Byte	0x02
3	Sequence	Byte	
4	Response code	Byte	Valid response codes: OK ERR_UNSUPPORTED_VERSION See Table 6 for response code descriptions
5	Version	Byte	
6	Manager sequence number	Byte	
7	Client sequence number	Byte	

response code—The response code is used by the manager to indicate the result of session establishment. These codes are defined in Table 6.

version—Protocol version supported by this manager. If the protocol version received in the *HELLO* packet is supported by the manager, *HELLO_RESPONSE* will contain a *response code* of 0x00 (OK). If the protocol version received in the *HELLO* packet is not supported by the manager, *HELLO_RESPONSE* will contain a response code of 0x01 (ERR_UNSUPPORTED_VERSION), and the version field will contain the supported version. The manager currently supports only Serial Protocol version 3.

manager sequence number—The *manager sequence number* establishes the sequence number of the manager reliable data stream. The first reliable message (Notification) sent by the manager should add one to the *manager sequence number* and use this value in the *sequence* field of the packet header. Subsequent reliable messages should increment the sequence number of the previous message.

client sequence number—The *client sequence number* confirms the client’s reliable sequence number received in *HELLO* message. The *client sequence number* also establishes the sequence number of the client command stream. The first reliable request packet (command) sent by the client should add one to the *client sequence number* and use this value in the *sequence* field of the packet header. Subsequent reliable request packets should increment the sequence number of the previous reliable request.

Table 6 Response Codes for HELLO_RESPONSE

Hex Value	Name	Description
0x00	OK	The session is successfully established.
0x01	ERR_UNSUPPORTED_VERSION	Unsupported protocol version.

Session Packet Processing

This section describes the communications link between the *client* and the *manager*. The serial API offers two modes of packet transmission: best-effort (low overhead) communication and reliable (packet-acknowledged) communication.

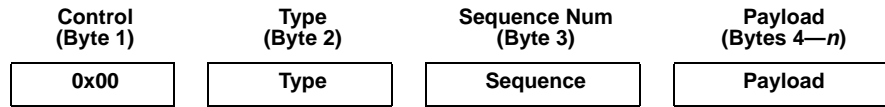
Note that communications between the *client* and the *network device* is discussed in “End-to-end Wireless Communication” on page 19.

Best-effort Communication (non-ACK)

Packets requiring no explicit receipt acknowledgement can be sent using a best-effort communication mechanism. This lowest-overhead method is best suited when the application layer is performing acknowledgements or when loss of data is tolerable. Applications requiring guaranteed delivery can subscribe to reliable data/event flow. At the receiver, all packets with the best-effort delivery flag (see “Control”) are forwarded to the application layer, and no ACKs are generated. Consequently, the sender receives no feedback about the success of individual packet delivery.

Best-effort Packets

In best-effort packets, the sequence number field is unused and set to zero.



The following diagram illustrates this principle:

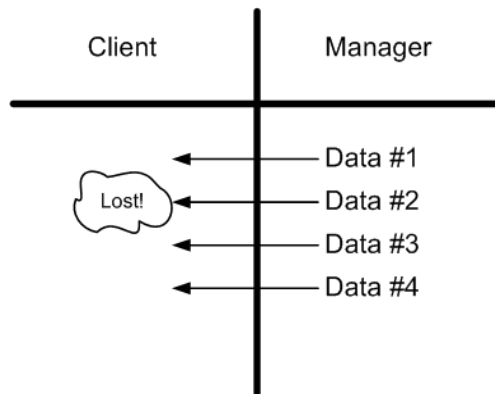


Figure 9 Best-effort Communication

Reliable Communication

Reliable communication should be used when guaranteed delivery of packets is required. Every packet marked by the sender as reliable (see “Control”) is explicitly acknowledged by the receiver. The sender must receive the acknowledgement for a reliable data packet before it may send the next reliable data packet.

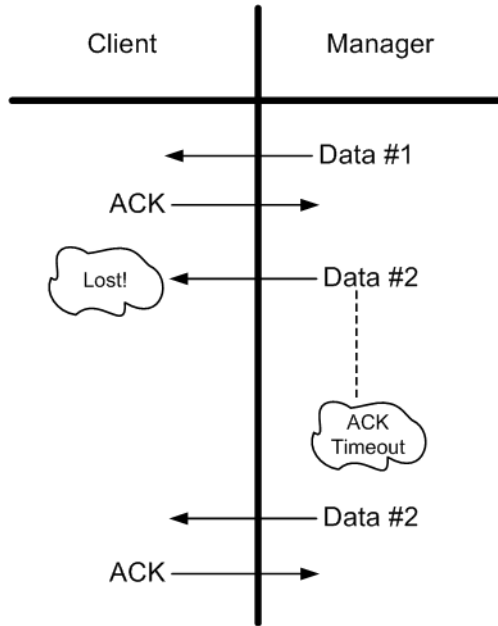


Figure 10 Reliable Communication

Reliable Packets

Each reliable packet sent must have the following format:

Control (Byte 1)	Type (Byte 2)	Sequence Num (Byte 3)	Payload (Bytes 4—n)
0x02	Type	Sequence	Payload

Byte	Description
Control byte	The ACK bit equals 0 and the SERVICE_TYPE equals 1, resulting in a Control byte value of 0x02
Type	This byte is different for each packet type. See the “Commands” on page 27 and “Notifications” on page 90 for details.
Sequence Num	A unique sequence number. This field is required for reliable communication.
Payload	Payload is optional, depending on packet type.

ACK Packets

Each ACK packet sent must have the following format:

Control (Byte 1)	Type (Byte 2)	Sequence Num (Byte 3)	Payload (Bytes 4– <i>n</i>)
0x03	Type	Sequence	Payload

Byte	Description
Control byte	The ACK bit equals 1 and the SERVICE_TYPE equals 1, resulting in a Control byte value of 0x03
Type	For packets requiring no application response (link-layer acknowledge only), this value should be 0. Otherwise, this value should be the type of the response packet.
Sequence Num	The sequence number being acknowledged.
Payload	Payload is optional, depending on packet type. See “Commands” on page 27 for information about specific packets.

Sequence Numbers

Only one acknowledged packet can be outstanding at a time, and the sender should not send the next packet (whether best-effort or reliable) until the previous packet is acknowledged. Thus, there are few requirements on how sequential numbers should be managed. The receiver merely stores the sequence number of the previous reliable packet and processes the next packet if its sequence number is different from the previous one. If the sequence number is the same, the receiver acknowledges the duplicate packet without processing it.

The sequence number received during the HELLO exchange is treated as the sequence number of previous packet, so the first reliable packet in the session should contain the next sequence number. The sequence number to begin a session may start at any value, but successive values must increment by one modulo 256. That is, in C syntax, the sequence number (*n*) should be:

$$n = (n + 1) \% 256.$$

Full-duplex Communication

The reliable flow is full-duplex, that is, both sides can send reliable data asynchronously from each other. Thus, while waiting for an ACK for a reliable packet, each device can receive a reliable packet from the other side, and should be ready to respond with an ACK.

Processing Guidelines

At the receiver, a packet received with a “reliable” flag (see “Control” on page 7) should be processed as follows:

- ❶ Compare sequence number of the packet with the previous sequence number of the sender.
- ❷ If the sequence numbers match, the packet is considered a duplicate. The receiver sends a copy of the previous reply and drops the packet.

If the numbers do not match, the packet is forwarded to the application layer. If an application-layer response is required, it is sent back in the acknowledgement packet. Otherwise, a packet with no application payload is sent back. A copy of last reply is always stored until the next reliable packet is received.

At the sender, the following procedure should be followed to send a reliable packet:

- ❶ Assign the next sequence number to the packet.
- ❷ Send the packet.
- ❸ Set timeout for ACK reception (1 second recommended).
- ❹ If an ACK with the correct sequence number is received, the packet transmission is complete. If the ACK contains payload for application data, forward it to the application.
- ❺ If the ACK timeout has been exceeded retry steps 2-5 (3 times recommended).
- ❻ If a maximum number of retries has been reached, the session should be terminated.

Example of Failed Communication

The following example illustrates what happens when a data packet or an ACK packet is lost during reliable data exchange.

- ❶ First, the *getTime* command with sequence number 5 is sent, and the manager responds with packet containing ACK and *getTime* response with sequence number 5.
- ❷ Next, the client sends the *sendRequest* request with sequence number 6, and the packet is lost. Following a timeout, the client resends the packet with sequence number 6. Now, the packet is received by the manager, but the response packet with sequence number 6 gets lost.
- ❸ Finally, the client retries sending the *sendRequest* request a third time with sequence number 6. The manager receives the duplicate packet, sends a copy of its response with sequence number 6, and the client receives it.

Error Handling

ACKs without application-level responses—Most reliable commands require application level responses to be sent as part of the ACK packet. For such commands, it is illegal for either side to send link-layer ACK containing no application response.

! **Important:** Receiving empty ACKs for commands requiring a specific application response can cause the link-layer and application layer to get out of sync. *Such behavior is considered an error and should cause session termination.*

Non-matching application responses—Either side expecting a packet with an application-level response should validate that the received packet type is the one it expects.

! **Important:** Receiving an invalid packet should cause session termination.

End-to-end Wireless Communication

Users may choose reliable or best-effort communication for serial transmissions between the client and the network device. The network manager supports end-to-end reliable communication in the *downstream* direction. Downstream communication refers to packets sent from the client to the network device.

Reliable Wireless Communication

In end-to-end reliable communication, packets sent wirelessly are acknowledged end-to-end, providing confirmation to the client application layer that the packet was successfully delivered. If the confirmation is not received, the manager will re-transmit the packet. With reliable communication, only one packet per destination address can be in transit in the network at a time. Additional packets are queued by the manager.

To send an end-to-end reliable request, the client uses the *sendRequest* command with the *isReliable* field set. When the manager receives the response from the network, it forwards it to the client using a *serial data* notification.

Applications requiring guaranteed delivery should use the end-to-end reliable communication mechanism.

Best-effort Wireless Communication

If the *isReliable* field not set in the *sendRequest* command sent by the client, no *serial data* notification is sent to the client. Therefore, the application receives no feedback about the success of individual packet delivery. Overall network power consumption is lower since there is no additional end-to-end acknowledgement. In addition, network throughput may be higher since multiple best-effort packet at a time may be in transit to a specific destination. Best-effort communication is best suited when no application acknowledgement is required or when a small percentage of lost packets is tolerable.

! **Important!** Broadcast packets (packets sent to all motes in the network) must always be sent using best-effort communication.

Network Bandwidth Control

IA-510 network managers support dynamic bandwidth to accommodate the bandwidth requirements of complex applications common in industrial monitoring and control. For example, an IA-510 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 IA-510 network 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 IA-510 manager API commands for both requesting bandwidth and controlling the bandwidth allocated to service requests. The IA-510 network manager API enables the following:

- Limits on Service-Requested bandwidth—defines maximum bandwidth that may be granted to service requests.
- Manager-requested bandwidth—defines network-wide minimum bandwidth.
- Allocated bandwidth—returns the total bandwidth allocated to a specific device.
- Pipe activation—enables 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 command 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 IA-510 Mote Serial API Guide*.

Table 7 Bandwidth Commands and Parameters

Manager API Parameter	Description
User-Settable Bandwidth Limits	
setNetwork command, minimum services packet period parameter	Limits non-pipe services. Defines maximum bandwidth (minimum data period) that a single mote may be allocated for the total of non-pipe, user requested bandwidth. Limits service requests from mote

Table 7 Bandwidth Commands and Parameters (Continued)

Manager API Parameter	Description
setNetwork command, minimum pipe packet period parameter	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.
Manager-API Driven Bandwidth Requests	
setNetwork command, requested base packet period parameter	Network-wide minimum bandwidth. Defines network-wide minimum bandwidth (maximum packet period). Note that the <i>setNetwork</i> command, requested base packet period parameter applies to manager-controlled bandwidth, which is independent from bandwidth requested through mote service requests.
Allocated Bandwidth	
getMote command, allocated packet period parameter	Total bandwidth allocated to a specific mote. Returns the total usable non-pipe bandwidth (1/allocated packet period) that was allocated to the mote. May be compared against <i>setNetwork</i> command, minimum services packet period limit. Includes bandwidth allocated from: <ul style="list-style-type: none"> • <i>setNetwork</i> command, requested base packet period • mote service requests
getMote command, allocated pipe packet period parameter	Pipe bandwidth. Returns usable bandwidth allocated to the pipe.
Service Denied notification	Serial API notification that indicates when a network service has been denied. The minimum services packet period parameter limit has been reached or there is a bottleneck in the network, as indicated by the needs neighbor field in the <i>getMote</i> command.
Pipe Activation	
activateFastPipe command	Serial API command to turn on pipe. Requests activation of a fast pipe to a specific mote. Parameters are MAC address and pipe direction. Valid values for pipe direction are as follows: 0x00 = Upstream 0x01 = Downstream 0x02 = Both upstream and downstream

Table 7 Bandwidth Commands and Parameters (Continued)

Manager API Parameter	Description
Pipe On network event	Serial API pipe activation notification. Notification sent by the network manager via the manager API, signifying that a pipe has been activated. The notification includes the MAC address of the mote at the end of the pipe.
getMote command, pipe status parameter	Status of pipe. Pipe states are as follows: 0x00 = Off 0x01 = Action pending 0x02 = On (bidirectional) 0x03 = On (upstream) 0x04 = On (downstream)

Network ID

The network ID is an identifier shared by the manager and the network motes, and serves to bind motes to the proper network. When a mote “hears” an advertisement with a network ID that matches its own, the mote is synchronized to its network manager and able to insert a join request into the network. Dust Networks’ network manager ships with a default network ID. To change the network ID, use the commands described in Table 11.

Note that a mote’s network ID can also be set via the mote serial API, as described in the *IA-510 Mote Serial API Guide*.

Table 11 API Commands for Changing the Network ID

Command	Description
setNetwork	Changes the network ID on the manager. For more information, see “setNetwork” on page 81.
exchangeNetworkId	Changes the network ID on the manager and all “Operational” network motes. For more information, see “exchangeNetworkId” on page 39.

Network Security

The IA-510 network manager *setSecurity* API command offers a choice of two network security modes—*Accept Common Join Key* and *Accept ACL*. These modes are described in the following sections.

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. Dust Networks’ network manager ships with a default common join key, which should be considered public knowledge. Therefore, it is highly recommended to change the common join key to a secret one. The default common join key is typically changed in advance of network deployment, but can also be changed after the network has formed. To change the common join key, use the commands described in Table 12.

Table 12 API Commands for Changing the Common Join Key

Command	Description
setSecurity	Changes the common join key on the manager. For more information, see “setSecurity” on page 84.
exchangeJoinKey	Changes the common join key on the manager and all “Operational” network motes. For more information, see “exchangeJoinKey” on page 37.

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 controller, which uses the *setAclDevice* command to add motes and the *deleteAclDevice* 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 *Mote Join* event notifications

that are sent to the manager when a mote joins the network (see “Mote Join” on page 102).

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 when in Accept ACL mode, use the commands described in Table 13.

Table 13 API Commands for Changing the Mote Join Key

Command	Description
setAcDevice	Changes a specific mote join key on the manager’s ACL list. For more information, see “setAcDevice” on page 76.
exchangeMoteJoinKey	Changes a specific mote join key on the manager’s ACL list as well as the network mote itself (assuming the mote is “Operational”). For more information, see “exchangeMoteJoinKey” on page 38.

Field Mask

Field mask is a bitmap field that indicates which fields in the packet payload contain valid information. *Field mask* is included in the response packet of *get* commands and in the request and response packets of *set* commands. A *field mask* bit is set to (1) to indicate a valid field and to (0) to indicate a field containing an invalid value. Fields are numbered in the order that they appear in the structure, counting from the field immediately following the *field mask*. The *field mask* is arranged in an array of four bytes, as indicated in Table 8.


 **Important!** With the exception of the *mote ID* and *MAC address* fields, all other read-only fields in the packet payload must have their field mask bit set to (0) or an error message is generated and the command will be ignored.

Table 8 Field Mask Array

Field Mask	Bit Mask Values							
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01
Byte [0]	Field 8	Field 7	Field 6	Field 5	Field 4	Field 3	Field 2	Field 1
Byte [1]	Field 16	Field 15	Field 14	Field 13	Field 12	Field 11	Field 10	Field 9
Byte [2]	Field 24	Field 23	Field 22	Field 21	Field 20	Field 19	Field 18	Field 17
Byte [3]	Field 32	Field 31	Field 30	Field 29	Field 28	Field 27	Field 26	Field 25

For example, to use the *setMote* command (see page 78) to set both the mote name (field 3) and enable routing (field 8), combine the bit masks (using bitwise-OR) to create the field mask, as follows:

Bit mask for field 3 = 0x04

Bit mask for field 8 = 0x80

Field mask byte [0] = 0x80 | 0x04 = 0x84

Field mask byte [1] = 0

Field mask byte [2] = 0

Field mask byte [3] = 0

Mote Joining

When a WirelessHART device joins the network the manager receives two WirelessHART commands, Command 0 and Command 20, as part of the *join* request. The manager forwards these commands to the OEM controller through the manager API as a *serial data* notification. For more information about the *serial data* notification, see “Serial Data Notification Packet” on page 92.

Response Codes

The following table describes the response codes that may be returned in the command response packet. The “Commands” on page 27 section specifies the response codes returned for each command.

Table 9 Response Codes

Hex Value	Error Message	Description
0x00	API_OK	The application layer has processed command successfully.
0x01	API_END_OF_LIST	The iteration has reached the end of the list of objects.
0x02	API_OBJECT_NOT_FOUND	The object was not found.
0x03	API_VAL_CROSSREF	Validation error. The cross reference failed.
0x04	API_VAL_ENUM	Validation error. The value was not in the enumeration.
0x05	API_VAL_PATTERN	Validation error. The value contained one of the following invalid characters: @, &, <, or >.
0x06	API_VAL_PATTERN_HEX	Validation error. The value contained an invalid hexadecimal character (valid characters are a-f, A-F, and 0-9).

Table 9 Response Codes (Continued)

Hex Value	Error Message	Description
0x07	API_VAL_ENCKEY	Validation error. The encryption key must contain exactly 32 hex characters.
0x08	API_VAL_DATAPACKET	Validation error. The message does not fit in the packet.
0x09	API_VAL_REQFIELD	Validation error. A required field is missing.
0x0A	API_VAL_LISTLEN	Validation error. Cannot add the new object because the limit has been reached.
0x0B	API_VAL_UNCHANGEABLE	Validation error. The element cannot be changed.
0x0C	API_NOTUNIQCMD	A similar command is already being processed.
0x0D	API_VAL_STATE	Validation error. The current state does not allow this action to be performed.
0x0E	API_VAL_MINMAX	Validation error. The maximum value must be greater than the minimum value.
0x0F	API_VAL_TIMEBOUNDARY	Validation error. The time is not on the required 15-minute boundary.
0x10	API_VAL_STARTEND	Validation error. The end time must be after start time.
0x11	API_VAL_LAST_ELEMENT	Validation error. Cannot delete the last element.
0x12	API_NETLAYER_FULL	The manager's transmit queue is full.
0x13	API_VAL_CANT_CREATE	Validation error. Cannot create the object because it already exists.
0x14	API_CLI_NULL_COMMAND	Empty CLI command.
0x15	API_CLI_WRITE_ERROR	Could not write to the CLI session.
0x16	API_INVALID_MOTE	The specified mote is invalid.
0x17	API_INVALID_COMMAND	The command is invalid.
0x18	API_INVALID_ARGUMENT	The argument to the command is invalid.
0xE8	API_GENERAL_ERROR	A general error that does not fall into any other error category.

Commands

This section provides detailed information for each serial API command. *All commands are immediately effective unless otherwise noted.* Commands consist of a reliable synchronous request message and a reliable response message. The packet contents are represented in a table format, with the fields listed in order of appearance and the Data Type column indicating the number of bytes in the field. For example, the request packet for *deleteAclEntry* command (Figure 14) contains a total of 11 bytes. Note that the packet header fields appear shaded and the packet payload is unshaded. Refer to the section “Data Types” on page 6 for details regarding the structure of each data type.

Table 10 deleteAclEntry Request

	Message Byte	Description	Data Type	Value
1 byte	1	Control	Byte	0x02
1 byte	2	Type	Byte	0x4D
1 byte	3	Sequence	Byte	
8 bytes	4-11	MAC address	Byte [8]	

Figure 14 Example Packet

Table 11 provides a short description of each command. Refer to the Appendices for example usages of the commands, including an overview of software update procedures.

Table 11 Command Summary

Command	Description
Connecting to and Disconnecting from Serial API	
HELLO HELLO_RESPONSE MGR_HELLO	Establishes a serial session with the manager.
disconnect	Shuts down the serial API server. The serial port can then be use for other applications.
Sending and Receiving Data	
sendRequest	Sends serial data to a specific mote.
sendResponse	Sends a response to a command or data packet received from the mote.
subscribe	Subscribes to notifications to receive data, network events, and alarms.
Manager/Network Settings and Descriptions	
adjustTime	Changes the time on the manager to the specified reference time. The adjustTime command requires that the NTP client is enabled and configured properly. Refer to Appendix E for more information.

Table 11 Command Summary (Continued)

Command	Description
getTime	Returns the manager's current UTC time and corresponding ASN (absolute slot number).
getSystem	Returns a general system description for the manager, including manager name, location, hardware model, hardware revision, software version and current time value.
setSystem	Sets the manager's system configuration parameters.
getNetwork	Returns a general network description that includes the network ID, maximum allowed number of motes, number of motes in network, and frame size.
setNetwork	Sets network configuration parameters, such as network ID, frame size, and optimization.
getMote	Returns the configuration for a given mote.
getNextMote	Used to iterate through the list of all motes in the network.
setMote	Sets configuration for a given mote.
deleteMote	Deletes a mote from the list of network motes maintained by the manager.
getPath	Returns the description for a given path.
getNextPath	Used to iterate through the list of paths for a given mote.
Security	
getSecurity	Returns the current network security settings.
setSecurity	Sets the network security settings.
getAclDevice	Returns the MAC address and join key for a given mote on the access control list (ACL).
getNextAclDevice	Returns the MAC address and join key for the next mote on the access control list (ACL).
setAclDevice	Adds a mote to the access control list (ACL).
deleteAclDevice	Removes a mote from the access control list (ACL).
exchangeNetworkId	Causes the manager to distribute a new network ID and perform a synchronized network switch-over to the new ID.
exchangeJoinKey	Causes the manager to distribute a new join key and perform a synchronized network switch-over to the new key.
exchangeMoteJoinKey	Triggers the manager to distribute a new join key to a specific mote.
exchangeNetworkKey	Causes the manager to generate a new network key, distribute it to motes, and perform a synchronized network switch-over to the new key.
exchangeSessionKey	Causes the manager to distribute a new random session key for a pair of motes.
getChannelBlacklist	Requests the list of frequencies on the channel blacklist.
setChannelBlacklist	Sets the channel blacklist.

Table 11 Command Summary (Continued)

Command	Description
Adding and Removing Devices	
activateAdvertising	Activates advertising on a mote.
decommissionDevice	Removes a mote from the network and from the manager's list of network motes and the access control list (ACL). The mote is made a leave node and is then ready to be powered down.
Network Performance	
getLatency	Returns estimated latency (in milliseconds) for a given mote.
getNetStatistics	Returns data reliability, data latency, and path stability statistics for a given period or for the lifetime of the network (lifetime average).
getMoteStatistics	Returns mote statistics for a given period or for the lifetime of the mote. Mote statistics include reliability, average latency, number of joins, voltage, charge consumption and temperature.
getPathStatistics	Returns path statistics for a given period or for the lifetime of the path. Path statistics include path stability and strength.
getPrevEvent	Returns information about the previous event.
getNextAlarm	Returns information about the next alarm in the current iteration.
getSLA	Returns the network service level agreement (SLA) thresholds for minimum network reliability, maximum network latency, and minimum path stability.
setSLA	Sets the network service level agreement (SLA) thresholds for minimum network reliability, maximum network latency, and minimum path stability.
Administrative Commands	
activateFastPipe	Activates a pipe between the manager/gateway and a mote.
deactivateFastPipe	Deactivates a pipe between the manager/gateway and a mote.
pingMote	Pings a specified mote.
reset	Resets a mote or the network, or clears statistics.
cli	Sends a cli command.
Mote Software Update Commands*	
getOtapFile	Gets the filename of the file specified by the index.
getOtapMote	Gets the OTAP information about the specified mote.
getNextOtapMote	Gets the OTAP information about the next OTAP mote.
startOtap	Triggers the manager to start an OTAP (over-the-air programming) session.
getOtapStatus	Returns the status of an OTAP (over-the-air programming) session in progress.
cancelOtap	Triggers the manager to cancel an OTAP (over-the-air programming) session that is in progress.
* Refer to Appendix D for an overview of software update procedures.	

activateAdvertising

Description

The *activateAdvertising* command activates advertising on all motes. The response code “OK” indicates that the manager has accepted the command and will attempt to turn on advertising. To determine the advertising status, wait a few seconds after issuing the *activateAdvertising* command and then issue the *getMote* command and check the advertising status field (see “getMote” on page 46).

Request Packet

Table 12 activateAdv Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x5F
3	Sequence	Byte	
4-11	MAC address	Byte[8]	0xFF FF FF FF FF FF FF FF (activates advertising on all motes)
12	Timeout	Byte	

MAC address—Packed as an array of bytes with the most significant byte first.

Timeout—The number of minutes (from 1 to 255) that advertising should remain on. After this period, advertising is turned off.

Response Packet

Table 13 activateAdv Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x60
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	MAC address	Byte[8]	

activateFastPipe

Description

This command activates a pipe between the manager/gateway and a mote (identified by its MAC address). Only one pipe may be active in the network at a time (see “End-to-end Wireless Communication” on page 19). A notification is sent when the pipe has been successfully turned on (see “Pipe On” on page 104). Another way to determine the pipe status is to issue the *getMote* command and check the pipe status field.

Note that the actual pipe bandwidth allocated by the manager depends on power information reported by the destination mote and the motes through which the pipe is built. To determine the allocated pipe bandwidth issue the *getMote* command and check the allocated pipe packet field.

Request Packet

Table 14 activateFastPipe Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x59
3	Sequence	Byte	
4-11	MAC address	Byte[8]	
12	Pipe direction	Byte	0x00 = Upstream 0x01 = Downstream 0x02 = Both upstream and downstream

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

Table 15 activateFastPipe Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x5B
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_ENUM API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.

cancelOtap

Description

The *cancelOtap* command triggers the manager to cancel an OTAP (over-the-air programming) process that is currently running.

- ! **Important:** Cancelling OTAP once the OTAP process has reached the commit state (see “getOtapStatus” on page 63) will result in motes running different software versions.

Request Packet

Table 16 cancelOtap Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x63
3	Sequence	Byte	

Response Packet

The OK (0x00) response code indicates that the application layer has processed the command correctly.

Table 17 cancelOtap Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x64
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_NETLAYER_FULL API_INVALID_COMMAND See Table 9 for response code descriptions.

cli

Description

This command triggers the manager to tunnel a given command line interface (CLI) command to manager’s command line interface. For more information, refer to the *SmartMesh IA-510 CLI Commands Guide*.

Request Packet

Table 18 cli Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x65
3	Sequence	Byte	
4-n	CLI command	Char[]	

For example, to tunnel the CLI command “exec activateFastPipe” to activate a pipe between the manager and a mote with MAC address of 00-00-00-00-00-40-C9 the request packet would contain the following information:

Control (Byte 1)	Type (Byte 2)	Sequence (Byte 3)	Payload (Bytes 4—45)
0x02	0x65	Sequence	exec activateFastPipe 00-00-00-00-00-40-C9

Response Packet

The manager’s response to the given CLI command is tunneled back to the client via *cli* notifications (see “CLI Notification Packet” on page 97). The OK (0x00) response code indicates that the manager accepted the command for processing.

Table 19 cli Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x66
3	Sequence	Byte	
4	Response code	Byte	API_OK API_CLI_NULL_COMMAND API_CLI_WRITE_ERROR See Table 9 for response code descriptions.

deactivateFastPipe

Description

This command deactivates a pipe between the manager/gateway and a mote (identified by its MAC address). Only one pipe may be active in the network at a time. A notification is sent when the pipe has been turned off successfully (see “Pipe Off” on page 104).

! **Important:** This command can only deactivate a pipe that was activated by the manager API. Do not call this command to deactivate a pipe that was activated as a result of a service request from a network device.

Request Packet

Table 20 DeactivateFastPipe Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x5A
3	Sequence	Byte	
4-11	MAC address	Byte[8]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet**Table 21 DeactivateFastPipe Response**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x5C
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.

decommissionDevice**Description**

The *decommissionDevice* command prepares a mote for removal from the network. The manager re-routes traffic to make the mote a leaf node (a node with no other motes reporting to it). When the mote is ready to be removed from the network, a *Mote Disconnect* event is generated. Advertising is not triggered when the mote is removed.

Response Packet

The argument to *decommissionDevice* is the MAC address of the mote.

Table 22 decommissionDevice Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x5D
3	Sequence	Byte	
4-11	MAC address	Byte[8]	

MAC address—Packed as an array of bytes with the most significant byte first.


Response Packet**Table 23 decommissionDevice Response**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x5E
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	MAC address	Byte[8]	

deleteAclDevice

Description

The *deleteAclDevice* command removes a mote from the access control list (ACL). After a mote is deleted from the ACL list it will not be able to rejoin the network the next time the mote resets. See also “deleteMote” on page 35.

 **Note:** When the security mode is set to “Accept ACL,” only the motes on the ACL are accepted into the network. When the security mode is set to “Accept Common Join Key,” motes are accepted into the network if they have the common join key or are listed on the ACL (see “setSecurity” on page 84).

Request Packet

The argument to *deleteAclDevice* is the MAC address of the mote.

Table 24 deleteAclDevice Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x4D
3	Sequence	Byte	
4-11	MAC address	Byte [8]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

The OK (0x00) response code indicates that the application layer has processed the command correctly.

Table 25 deleteAclDevice Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x4E
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	MAC address	Byte [8]	

deleteMote

Description

The manager maintains a list of motes that are in the network or are expected to be in the network. The *deleteMote* command deletes a mote from the manager’s list and from the ACL. This ensures that the correct number of motes are counted against the system’s maximum mote limit. Use this command when a mote is no longer in the network and is not expected to be re-deployed. This command can only be issued for a mote in the Disconnecting state. To get state information for a mote use the *getMote* command described on page 39.

Request Packet

The argument to *deleteMote* is the MAC address of the mote.

Table 26 deleteMote Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x23
3	Sequence	Byte	
4-11	MAC address	Byte[8]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

The OK (0x00) response indicates that the application layer has processed the command correctly. The response includes the MAC address of the mote that is being deleted.

Table 27 deleteMote Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x24
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_STATE API_NETLAYER_FULL API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	MAC address	Byte[8]	

disconnect**Description**

The *disconnect* command closes the connection to the serial API server running on the manager. The serial port can then be used for other applications (see “Serial Port Process Switch” on page 2).

Request Packet**Table 28 disconnect Request**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x69
3	Sequence	Byte	

Response Packet

There is no response to the disconnect command.

exchangeJoinKey

Description

The *exchangeJoinKey* command triggers the manager to initiate a change of common join key for all motes in the network. 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. The *exchangeJoinKey* command cannot be used if the manager is in Accept ACL mode or if it is in Accept Common Join Key mode and there are entries in the access control list (ACL). For more information about the security modes, see “Network Security” on page 23.

The join key is exchanged without loss of data, and no resetting of motes is needed. 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 *Command Finished* event (see “Command Finished” on page 101) when the command is synchronously executed by the motes. The delay is typically tens of minutes after the *exchangeJoinKey* command is issued and depends on the network size and configuration. For networks using the P1 configuration, the delay is around 21 minutes. During this period, no additional *exchangeJoinKey* commands may be issued. Motes that reset during the exchange may or may not receive the new join key. If a mote does not receive the new join key it cannot re-join the network.

Note that network motes must be in the Operational state (see “getMote” on page 46) when the *exchangeJoinKey* command is issued in order for them to receive the new join key. For best results, use the following procedure:

- ❶ Before exchanging the join key, first determine if all network motes are operational by issuing the *getMote* command (see “getMote” on page 46). 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).
- ❷ When all motes are operational, issue the *exchangeJoinKey* command.
- ❸ When the *Command Finished* event is received, reissue the *getMote* command to determine if all motes are operational. Do one of the following depending on the outcome:
 - If all motes are operational, they have all received the new join key and no further action is needed.
 - 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 join key so that these motes can rejoin. Then repeat steps 1-3 with the new join key.

Request Packet

Table 29 exchangeJoinKey Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x1F
3	Sequence	Byte	
4-19	New key	Byte[16]	

Response Packet

The OK (0x00) response code indicates that the application layer has processed the command correctly.

Table 30 exchangeJoinKey Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x20
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_ENCKEY API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Callback ID	LongInt	

exchangeMoteJoinKey

Description

The *exchangeMoteJoinKey* command triggers the manager to initiate the exchange of the join key for a specific mote (identified by its MAC address) and update the access control list (ACL). 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 *Command Finished* event is generated (see “Command Finished” on page 101). The join key is exchanged without loss of data, and the mote does not need to be reset.

! **Important:** The time required to complete the command and issue a *Command Finished* event 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 (see “getMote” on page 46) 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 issuing the *getMote* command (see “getMote” on page 46). 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 *Command Finished* event is received, reissue the *getMote* command to determine if the mote is still operational. Do one of the following depending on the outcome:
 - If the mote is operational, it has received the new join key and no further action is needed.
 - If the mote is non-operational, wait up to an hour to see if it can rejoin the network. If it does not rejoin, repeat steps 2 and 3 with the old join key so that it can rejoin. Then repeat steps 1-3 with the new network join key.

Request Packet**Table 31** exchangeMoteJoinKey Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x51
3	Sequence	Byte	
4-11	MAC address	Byte[8]	
12-27	New key	Byte[16]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

The OK (0x00) response code indicates that the application layer has processed the command correctly.

Table 32 exchangeMoteJoinKey Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x52
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_ENCKEY API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Callback ID	LongInt	
9-16	MAC address	Byte[8]	

exchangeNetworkId**Description**

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 *Command Finished* event (see “Command Finished” on page 101) 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 around 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 (see Table 47) when the *exchangeNetworkId* command is issued in order for them to receive the new network ID. For best results, follow this process:

- ❶ Before exchanging the network ID, first determine if all network motes are operational by issuing the *getMote* command (see “getMote” on page 46). 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).
- ❷ When all motes are operational, issue the *exchangeNetworkId* command.
- ❸ When the *Command Finished* event is received, reissue the *getMote* command to determine if all motes are operational. Do one of the following, depending on the outcome:
 - If all motes are operational, they have all received the new network ID. Reset the network.
 - 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.

Request Packet

Table 33 *exchangeNetworkId* Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x21
3	Sequence	Byte	
4-5	New ID	ShortInt	

Response Packet

The OK (0x00) response code indicates that the application layer has processed the command correctly.


Table 34 *exchangeNetworkId* Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x22
3	Sequence	Byte	
4	Response code	Byte	API_OK API_VAL_UNCHANGEABLE API_VAL_MINMAX API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Callback ID	LongInt	

exchangeNetworkKey

Description

The *exchangeNetworkKey* command triggers the manager to generate a new network key and distribute it to the motes. The network key is exchanged without loss of data and no resetting of motes is needed. 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 *Command Finished* event (see “Command Finished” on page 101) when the command is synchronously executed by the motes. The delay is typically tens of minutes after the *exchangeNetworkKey* command is issued and depends on the network size and configuration. For networks using the P1 configuration, the delay is around 21 minutes. During this period, no additional *exchangeNetworkKey* commands may be issued. Note that if a mote resets during the exchange, it will receive the new network key when it re-joins the network.

 **Note:** The network key is a network-wide symmetric authentication key that is shared by the manager and the motes in its network. The network key is used to authenticate all messages other than join requests, activation messages, and advertisements on the data link (MAC) layer. The manager gives the network key to motes when they join the network.

Request Packet

Table 35 exchangeNetworkKey Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x1D
3	Sequence	Byte	

Response Packet

The OK (0x00) response code indicates that the application layer has processed the command correctly.

Table 36 exchangeNetworkKey Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x1E
3	Sequence	Byte	
4	Response code	Byte	API_OK API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Callback ID	LongInt	

exchangeSessionKey

Description

The *exchangeSessionKey* command triggers the manager to distribute a new random session key between the manager (or gateway) and a mote. The session key is used to encrypt all messages exchanged between the manager (or gateway) and a mote. When the session key exchange is completed, a *Command Finished* event is generated (see “Command Finished” on page 101). The session key is exchanged without loss of data and no resetting of motes is needed.

! **Important:** The time required to complete the command and issue a *Command Finished* event is may vary depending on the size and type of the network. During this period no additional *exchangeSessionKey* commands may be issued.

Request Packet

The MAC address is packed as an array of bytes with the most significant byte first.

Table 37 exchangeSessionKey Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x53
3	Sequence	Byte	
4-11	MAC address of manager (gateway)	Byte [8]	
12-19	MAC address of mote	Byte [8]	

Response Packet

The OK (0x00) response code indicates that the application layer has processed the command correctly.


Table 38 exchangeSessionKey Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x54
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_UNCHANGEABLE API_NETLAYER_FULL API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Callback ID	LongInt	
9-16	MAC address of manager (gateway)	Byte [8]	
17-24	MAC address of mote	Byte [8]	

getAclDevice

Description

The *getAclDevice* command returns the MAC address and join key for a specified mote on the access control list (ACL). The ACL contains the motes that are allowed to join the network when security is set to “Accept ACL.”

 **Note:** The join key is a symmetric encryption key that is shared between the manager the motes in its network. The join key is used by the manager and motes to encrypt and decrypt the join messages they exchange when the mote is attempting to join the network.

Request Packet

The argument to *getAclDevice* is the MAC address of the mote.

Table 39 getAclDevice Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x3B
3	Sequence	Byte	
4-11	MAC address	Byte [8]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

Table 40 getAclDevice Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x3C
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-16	MAC address	Byte[8]	
17-49	Join key	Char[33]	

field mask—Describe in “Field Mask” on page 24.

Join key—Note that the join key is represented as a string. For example, a join key beginning with the hexadecimal numbers “0x01 0x02 0x03...” would be represented as “010203...” in the join key field.

getChannelBlacklist

Description

The *getChannelBlacklist* command gets information about the channels that are blacklisted for the network. Blacklisted channels are not used in frequency hopping.

Request Packet

Table 41 getChannelBlacklist Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x31
3	Sequence	Byte	

Response Packet

Table 42 getChannelBlacklist Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x32
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-30	Frequency	ShortInt[11]	

field mask—Described in “Field Mask” on page 24.

frequency—Array containing the blacklisted frequencies. The elements of the frequency array must be a value from 2405 to 2480 (inclusive, incrementing by 5). The array is always 11 elements long (at most 11 channels may be blacklisted). Unused elements must be set to 0 if there are fewer than 11 channels in the blacklist. Only an odd number of channels may be blacklisted. Setting a new set of blacklist channels completely replaces the previous value.

getLatency

Description

The *getLatency* command returns the latency estimates (in milliseconds) for a given mote.

Request Packet

The argument to the *getLatency* command is the MAC address of the mote.

Table 43 getLatency Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x19
3	Sequence	Byte	
4-11	MAC address	Byte[8]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

The mote latency response contains a structure describing the round trip latency to a specific mote. The latency calculations take into account long-term bandwidth, such as bandwidth set by the manager API or requested by service requests, but not short-term bandwidth or pipe bandwidth.

Table 44 getLatency Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x1A
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	MAC address	Byte[8]	
13-16	Downstream latency	LongInt	
17-20	Upstream latency	LongInt	

MAC address—Packed as an array of bytes with the most significant byte first.

Downstream latency—The estimated upper limit of the expected downstream latency (in milliseconds) from the manager to that mote.

Upstream latency—The estimated upper limit of the expected upstream latency (in milliseconds) from the mote to the manager.

getMote

Description

The *getMote* command returns the description for a given mote (specified by its MAC address).

Request Packet

The argument to *getMote* is the MAC address of the mote.

Table 45 getMote Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x13
3	Sequence	Byte	
4-11	MAC address	Byte[8]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

Table 46 getMote Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x14
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-10	Mote ID	ShortInt	
11-18	MAC address	Byte[8]	
19-35	Name	Char[17]	
36	Power source	Byte	0x00 = Line 0x01 = Battery 0x02 = Recharge/power scavenging
37-38	Discharge current	ShortInt	
39-42	Discharge time	LongInt	
43-46	Recovery time	LongInt	
47	Enable routing	Byte	0x00 = Off 0x01 = On
48-64	Product name	Char[17]	
65-81	Hardware model	Char[17]	
82-98	Hardware revision	Char[17]	

Table 46 getMote Response (Continued)

Message Byte	Description	Data Type	Value
99-115	Software revision	Char[17]	While the mote is starting up, the version number is initially populated with the placeholder value "0.0.0-0" until the mote becomes operational and reports its value through the network (this may take a few minutes).
116	isAccessPoint	Byte	0x00 = Not an access point 0x01 = Access point
117-118	NumJoins	ShortInt	
119	State	Byte	See Table 47
120	Mote state reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error
121-128	Join time	DustTime	
129-132	Voltage	FixedPoint	
133-134	Number of neighbors	ShortInt	
135-138	Allocated packet period	LongInt	
139-142	Allocated pipe packet period	LongInt	
143	Pipe status	Byte	0x00 = Off 0x01 = Action pending 0x02 = On (bidirectional) 0x03 = On (upstream) 0x04 = On (downstream)
144	Advertising status	Byte	0x00 = On 0x01 = Off 0x02 = Action pending
145	Needs neighbors	Byte	0x00 = False 0x01 = True

The mote description contains a structure describing the current state of a single mote. *field mask*—Described in “Field Mask” on page 24.

mote ID—A unique identifier used for iterating through the list of all motes in the network. Since the mote ID is assigned to a mote when it joins the network, this identifier may change from join to join.

MAC address—The IEEE unique address of the mote. This address is typically stored persistently on the mote.

power source—Indicates the mote’s power source (battery, line, recharge/ power scavenging).

discharge current, *discharge time*, and *recovery time*—Collectively used to describe the current sourcing capabilities of the three possible types of power supply feeding the mote.

- The discharge current indicates the discharge current, in μA .
- The discharge time is the maximum time (in seconds) that the power supply can sustain the discharge current before experiencing a voltage drop.
- The recovery time is the time (in seconds) required by the power supply to recover from a power drain (for example, recharge its capacitors).

enable routing—Indicates whether the mote may be configured for routing data from other motes.

isAccessPoint—Indicates that this mote is an access point. An access point mote bridges the wireless mesh network to a larger data network (for example, an Ethernet or RS-485 network) and may physically reside in a gateway.

numjoins—Indicates the total number of times that the mote has joined the network since the last time the manager reset.

join time—Indicates the last time that the mote joined the manager. If the mote has not joined since the manager started, the join time will be zero.

state—See Table 47.

voltage—A fixed point value containing the reading from the last battery measurement.

number of neighbors—The number of motes within range of this mote, both currently and potentially connected.

allocated packet period—The currently allocated bandwidth (in msec/packet) from mote to gateway.

allocated pipe packet period—The currently allocated bandwidth (in msec/packet) for the pipe. The value 0x01 (action pending) means the manager is in the process of changing the state of the pipe from on to off (or vice-versa).

pipe status—Indicates the status of the pipe at the mote.

advertising status—Indicates the status of advertising at the mote.

needs neighbors—Indicates whether a mote has only one parent and (or) has insufficient links. A well formed network should have only one mote (the “single-parent” mote in the first hop) with this flag on. If the flag is on for other motes, it indicates that an additional mote needs to be added near them.

Mote States

Table 47 Mote States

Value	Mote State	Description
0x00	IDLE	Mote has not been part of the network since the manager started.
0x01	LOST	Mote is not currently part of the network.
0x02	NEGOTIATING1	Mote is in the process of joining the network.
0x03	NEGOTIATING2	Mote is in the process of joining the network.
0x04	CONNECTED	Mote is connected to the network.
0x05	OPERATIONAL	Mote is operational.
0x06	DISCONNECTING	Mote may be physically removed from network without affecting network.

getMoteStatistics

Description

The *getMoteStatistics* command returns a set of mote statistics. Statistics may be requested for the current 15-minute period (on the quarter hour), a specific 15-minute period, a specific day, or for the lifetime of the mote (lifetime average).

Request Packet

Table 48 getMoteStatistics Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x29
3	Sequence	Byte	
4-11	MAC address	Byte [8]	
12	Statistics period	Byte	0x00 = Current 15-minute period 0x01 = Specific 15-minute 0x02 = Specific day 0x03 = Lifetime statistics
13	Period index	Byte	0x00 through 0x5F

MAC address—Packed as an array of bytes with the most significant byte first.

statistics period and *period index*—Arguments used to specify the 15-minute period or the day for which statistics are requested:

- When specifying a 15-minute period, an index of 0 corresponds to the 15-minute period immediately preceding the current time. An index of 1 corresponds to the period before that one, and so on.
- When specifying a day, an index of 0 corresponds to the current day. An index of 1 corresponds to the day before that one, and so on.

For example, if you are using the index to specify a 15-minute period and it is currently 03:23, index 0 corresponds to the period from 03:00 to 03:15, and index 5 corresponds to the period from 01:45 to 02:00. The manager can store up to ninety-six 15-minute periods, so the index can range from 0 to 95.

If current or lifetime statistics are requested, the index is ignored.

Response Packet

Table 49 getMoteStatistics Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x2A
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-12	Requested index	LongInt	
13-20	Start time of interval	DustTime	
21-24	Reliability	FixedPoint	
25-28	Latency	LongInt	
29-30	NumJoins	ShortInt	
31-34	Voltage	FixedPoint	
35-38	Charge consumption	LongInt	
39-42	Temperature	FixedPoint	

The mote statistics response contains a structure describing the mote statistics for the requested time period.

field mask—Described in “Field Mask” on page 24.

requested index—The 15-minute period or day from the request packet.

start time of interval—The start time of the 15-minute period requested.

reliability—The percentage of generated packets that were received by the manager (0-100).

latency—The average time (in milliseconds) taken for packets generated at the mote to reach the manager.

num joins—Number of of times the mote joined the network during the requested time period.

voltage—The most recent battery reading that was taken during the requested time period.

charge consumption—The cumulative millicoulombs consumed by mote. Charge calculations are based on nominal part performance.

temperature—The temperature of the mote, in Celsius.

getNetStatistics

Description

The *getNetStatistics* command returns a set of network statistics. Statistics may be requested for the current 15-minute period (on the quarter hour), a specific 15-minute period, a specific day, or for the lifetime of the mote (lifetime average).

Request Packet

Table 50 getNetStatistics Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x27
3	Sequence	Byte	
4	Statistics period	Byte	0x00 = Current 15-minute period 0x01 = Specific 15-minute 0x02 = Specific day 0x03 = Lifetime statistics
5	15-minute period index	Byte	0x00 through 0x5F

statistics period, index—Arguments used to specify the 15-minute period or the day for which statistics are requested.

- When specifying a 15-minute period, an index of 0 corresponds to the 15-minute period immediately preceding the current time. An index of 1 corresponds to the period before that one, and so on.
- When specifying a day, an index of 0 corresponds to the current day. An index of 1 corresponds to the day before that one, and so on.

For example, if you are using the index to specify a 15-minute period and it is currently 03:23, index 0 corresponds to the period from 03:00 to 03:15, and index 5 corresponds to the period from 01:45 to 02:00. The manager can store up to ninety-six 15-minute periods, so the index can range from 0 to 95. If current or lifetime statistics are requested, the index is ignored.

Response Packet

Table 51 getNetStatistics Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x28
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-12	Requested index	LongInt	0x00 through 0x5F
13-20	Start time of interval	DustTime	

Table 51 getNetStatistics Response (Continued)

Message Byte	Description	Data Type	Value
21-24	Reliability	FixedPoint	
25-28	Stability	FixedPoint	
29-32	Latency	LongInt	
33-36	Lost upstream packets	LongInt	

The network statistics response contains a structure describing the network statistics for the requested time period.

field mask—Described in “Field Mask” on page 24.

requested index—The 15-minute period from the request packet.

start time of interval—The start time of the 15-minute period requested.

reliability—The percentage of generated packets that were received by the manager (0-100).

stability—The ratio of successful packet transmissions to the total number of packet transmissions on all paths in the network (0-100).

latency—The average time (in milliseconds) taken for packets generated at the motes to reach the manager.

lost upstream packet—The total number of lost upstream packets across all devices and sessions. This field is only valid for lifetime statistics; it is not used for other statistics periods.

getNetwork

Description

The *getNetwork* command returns general information about the network, including the network ID, and number of motes in the network.

Request Packet

Table 52 getNetwork Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x11
3	Sequence	Byte	

Response Packet**Table 53** getNetwork Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x12
3	Sequence	Byte	
4	Response code	Byte	API_OK API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-25	Network name	Char[17]	
26-27	Network ID	ShortInt	
28	Optimization	Byte	0x00 = Off 0x01 = On
29-36	Maintenance start time	DustTime	
37-44	Maintenance end time	DustTime	
45-46	Max motes	ShortInt	
47-48	Number of motes	ShortInt	
49	Access point power amplifier	Byte	0x00 = Off 0x01 = On
50	Clear channel access	Byte	0x00 = Off 0x01 = On
51-54	Requested base packet period	LongInt	
55-58	Minimum services packet period	LongInt	
59-62	Minimum pipe packet period	LongInt	
63	Bandwidth profile	Byte	See Table 54.
64-67	Manual upstream frame size	LongInt	
68-71	Manual downstream frame size	LongInt	
72-75	Manual advertising frame size	LongInt	
76-77	Network queue size	ShortInt	
78-79	User queue size	ShortInt	

The response packet contains several configuration parameters used by the mote network.

field mask—Described in “Field Mask” on page 24.

network name—A user readable text field to describe the specific network installation.

network ID—An identifier that binds motes to the proper network. This value must be the same for all motes in the network and for the manager.

optimization—Indicates whether algorithms are enabled on the manager to continuously optimize network links to improve network performance. Dust Networks strongly recommends that optimization remain on (enabled) at all times.

maintenance start time, maintenance end time—These fields indicate a scheduled network maintenance period during which statistics-gathering will be disabled.

max motes—The maximum number of motes (excluding gateway motes, but including other motes in all states) that are allowed in the network.

number of motes—The number of motes (excluding gateway motes) that are in the “Live” state.

access point power amplifier—Indicates the RF output power setting of the access point mote (power amplifier is on or off).

clear channel access—Indicates whether network motes (including the access point mote) listen on a channel before they transmit.

requested base packet period (msec/packet)—Defines the base bandwidth that the manager should allocate for each mote. Note that requested base packet period does not include bandwidth requested through mote service requests and cannot be used to allocate bandwidth for services.

minimum services packet period—(msec/packet) Defines the maximum bandwidth that a single mote may be allocated for total non-pipe user requested bandwidth. This limits service requests from motes.

minimum pipe packet period—(msec/packet) Limits bandwidth on all pipes (manager-API requested pipes and pipes requested in service requests).

bandwidth profile—Determines the frame size (number of timeslots) for upstream, downstream, and advertising traffic. There are three profiles available: a normal profile (P1), a low-power profile (P2), and a manual profile. The manual profile lets you use the next three bytes to manually set the number of timeslots for upstream, downstream, and advertising traffic. For more information about bandwidth profile, see “Bandwidth Profiles” on page 55.

! **Caution:** The manual bandwidth profile is an advanced feature to be used only with direct support from Dust Networks applications engineering.

manual upstream frame size, manual downstream frame size, manual advertising frame size—Manually indicate the frame size (number of timeslots) for upstream traffic, downstream traffic, and advertising. These parameters are only applicable if the bandwidth profile is set to “manual.”

network queue size—The number of messages in the manager’s network management queue. The user queue size is the number of messages in the manager’s user command queue.

Bandwidth Profiles

The new bandwidth profile setting takes effect the next time the network reforms. After changing the network bandwidth, use the reset command (see page 72) to reset the network and cause it to reform.


Table 54 Bandwidth Profiles

Value	Profile Name	Description
0x00	Manual	This profile allows you to manually change the frame size (number of timeslots) for upstream, downstream and advertising traffic. ! Caution: The manual bandwidth profile is an advanced feature to be used only with direct support from Dust Networks applications engineering.
0x01	P1	This is the normal bandwidth profile setting and provides the following frame sizes: Upstream frame size = 1024 timeslots Downstream frame size = 256 timeslots Advertising frame size = 128 timeslots
0x02	P2	This is the low-power bandwidth profile setting and provides the following frame sizes: Upstream frame size = 1024 timeslots Downstream frame size = 2048 timeslots Advertising frame size = 128 timeslots

getNextAclDevice

Description

The *getNextAclDevice* command returns the MAC address and join key for the next mote on the access control list (ACL) following the mote you specify. Specifying MAC address 00-00-00-00-00-00-00-00-00 retrieves the first entry in the ACL list. The ACL contains the motes that are allowed to join the network when the security mode is set to “Accept ACL.”

 **Note:** The join key is a symmetric encryption key that is shared between the manager and the motes in its network. The join key is used by the manager and motes to encrypt and decrypt the join messages they exchange when the mote is attempting to join the network.

Request Packet

The argument to *getNextAclDevice* is the MAC address of the mote.

Table 55 getNextAclDevice Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x3D
3	Sequence	Byte	
4-11	MAC address	Byte [8]	

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet**Table 56** getNextACL Device Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x3E
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-16	MAC address	Byte[8]	
17-49	Join key	Char[33]	

field mask—Described in “Field Mask” on page 24.

getNextAlarm

Description

The *getNextAlarm* command returns information about the next alarm in the current iteration. This can be used to iterate through all the open alarms in the network.

Request Packet**Table 57** getNextAlarm Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x2D
3	Sequence	Byte	
4-7	Event ID	LongInt	

Response Packet**Table 58** getNextAlarm Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x2E
3	Sequence	Byte	
4	Response code	Byte	API_OK API_END_OF_LIST API_INVALID_COMMAND See Table 9 for response code descriptions
5-12	Packet timestamp	DustTime	
13-16	Alarm ID	LongInt	
17	Alarm type	Byte	See Table 59
18-n	Alarm data		

Alarm Types

Table 59 Alarm Types

Alarm Type	Description
0x00	Alarm Close
0x01	Mote Down
0x02	Low Reliability
0x03	High Latency
0x04	Low Stability
0x05	Reserved
0x06	Maximum Number of Motes Reached

getNextMote

Description

The *getNextMote* command returns information about the next mote in the list of motes.

Request Packet

The *getNextMote* command takes a single argument, the mote identifier for the last retrieved mote. Calling *getNextMote* with the identifier zero will always return the gateway mote as the first element of the iteration.

Table 60 getNextMote Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x15
3	Sequence	Byte	
4-7	Mote ID	LongInt	

Response Packet

Table 61 getNextMote Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x16
3	Sequence	Byte	
4	Response code	Byte	API_OK API_END_OF_LIST API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-10	Mote ID	ShortInt	
11-18	MAC address	Byte[8]	
19-35	Name	Char[17]	

Table 61 getNextMote Response (Continued)

Message Byte	Description	Data Type	Value
36	Power source	Byte	0x00 = Line 0x01 = Battery 0x02 = Recharge/power scavenging
37-38	Discharge current	ShortInt	
39-42	Discharge time	LongInt	
43-46	Recovery time	LongInt	
47	Enable routing	Byte	0x00 = Off 0x01 = On
48-64	Product name	Char[17]	
65-81	Hardware model	Char[17]	
82-98	Hardware revision	Char[17]	
99-115	Software revision	Char[17]	While the mote is starting up, the version number is initially populated with the placeholder value "0.0.0-0" until the mote becomes operational and reports its value through the network (this may take a few minutes).
116	isAccessPoint	Byte	0x00 = Not an access point 0x01 = Access point
117-118	NumJoins	ShortInt	
119	State	Byte	See Table 47
120	Mote state reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error
121-128	Join time	DustTime	
129-132	Voltage	FixedPoint	
133-134	Number of neighbors	ShortInt	
135-138	Allocated packet period	LongInt	
139-142	Allocated pipe packet period	LongInt	
143	Pipe status	Byte	0x00 = Off 0x01 = Action pending 0x02 = On (bidirectional) 0x03 = On (upstream) 0x04 = On (downstream)
144	Advertising status	Byte	0x00 = On 0x01 = Off 0x02 = Action pending
145	Needs neighbors	Byte	0x00 = False 0x01 = True

For field descriptions, see “getMote” on page 46.

getNextOtapMote

Description

The *getNextOtapMote* command gets OTAP information about the next OTAP mote. Specifying a mote ID of “0” returns the first OTAP mote in the list. See also “getOtapMote” on page 61.

Request Packet

Table 62 getNextOtapMote Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x6E
3	Sequence	Byte	
4-7	Mote ID	LongInt	

Response Packet

Table 63 getNextOtapMote Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x6F
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-10	Mote ID	ShortInt	
11-18	MAC address	Byte[8]	
19-22	Percent complete	FixedPoint	
23	Mote OTAP state	Byte	See Table 70

field mask—Described in “Field Mask” on page 24.

mote ID—A unique identifier used for iterating through the list of all motes in the network. Since the *mote ID* s assigned to a mote when it joins the network, this identifier may change from join to join.

MAC address—The IEEE unique address of the mote. This address is typically stored persistently on the mote.

percent complete—Percentage of OTAP completed on this mote.

mote OTAP state—See Table 70.

getNextPath

Description

The *getNextPath* command returns a response containing a list of structures indicating the connections to neighboring motes. See also “getPath” on page 64.

Request Packet

The command takes a set of arguments to indicate the mote and path.

Table 64 getNextPath Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x17
3	Sequence	Byte	
4-11	MAC address	Byte[8]	
12-15	Path ID	LongInt	0x00 = First path in the list

MAC address—Packed as an array of bytes with the most significant byte first.

path ID—The ID of the path. Using 0 as the path ID will return the first path in the list.

Response Packet

Field mask is described in “Field Mask” on page 24. The number of links indicates how many links comprise the path.

Table 65 getNextPath Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x18
3	Sequence	Byte	
4	Response code	Byte	API_OK API_END_OF_LIST API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-12	Path ID	LongInt	
13-20	Mote A MAC address	Byte[8]	
21-28	Mote B MAC address	Byte[8]	
29-30	Number of links	ShortInt	

getOtapFile

Description

The *getOtapFile* command gets the filename of the OTAP file (specified by the index) in the manager’s /root/otap directory. An index of 0x00 gets the first filename in the /root/otap directory.

Request Packet**Table 66** getOtapFile Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x6A
3	Sequence	Byte	
4	Index	Byte	0x00 = First filename in the /root/otap directory.

Response Packet**Table 67** getOtapFile Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x6B
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-41	Filename	Char[33]	

field mask—Described in “Field Mask” on page 24.

getOtapMote

Description

The *getOtapMote* command gets the OTAP information about the first mote in the list of motes that are receiving an OTAP. Use the *getNextOtapMote* command to request information about additional motes.

Request Packet**Table 68** getOtapMote Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x6C
3	Sequence	Byte	

Response Packet**Table 69** getOtapMote Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x6D
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-10	Mote ID	ShortInt	
11-18	MAC address	Byte[8]	
19-22	Percent complete	FixedPoint	
23	Mote OTAP state	Byte	See Table 70

field mask—Described in “Field Mask” on page 24.

mote ID—A unique identifier used for iterating through the list of all motes in the network. Since the mote ID is assigned to a mote when it joins the network, this identifier may change from join to join.

MAC address—The IEEE unique address of the mote. This address is typically stored persistently on the mote. The MAC address is packed as an array of bytes with the most significant byte first.

percent complete—Percentage of OTAP completed on this mote.

mote OTAP state—See Table 70.

Mote OTAP State**Table 70** Mote OTAP States

Value	Mote OTAP State	Description
0x00	NOT IN OTAP	Mote is not receiving OTAP at this moment because the OTAP file does not apply to this mote.
0x01	OTAP IN PROGRESS	OTAP is in progress on this mote.
0x02	OTAP CANCELLED	OTAP was cancelled for this mote.
0x03	LOCKED OUT	OTAP will not be performed on this mote because OTAP lockout was set for this mote.
0x04	LOW BATTERY	Mote battery power is too low for OTAP to be performed.
0x05	NO SPACE	Unexpected error. There is not enough space in the mote flash for the OTAP file.
0x06	FILE ERROR	Unexpected error. The OTAP file is invalid or cannot be recognized by the mote.
0x07	MISC. ERROR	Unexpected internal error occurred.
0x08	FINISHED OK	Otap was completed on this mote.

getOtapStatus

Description

The *getOtapStatus* command returns the status of an OTAP (over-the-air programming) session in progress.

Request Packet

Table 71 getOtapStatus Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x33
3	Sequence	Byte	

Response Packet

Table 72 getOtapStatus Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x34
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9	OTAP state	Byte	0x00 = None 0x01 = Initializing 0x02 = Upload 0x03 = Commit 0x04 = Completed
10-17	Elapsed time since OTAP started	DustTime	
18-19	Total number of motes in the network	ShortInt	
20-21	Number of motes that have confirmed receipt of OTAP files	ShortInt	
22-54	Current file	Char[33]	
55-56	Total number of retries for the current file	ShortInt	
57-58	Current retry for the current file	ShortInt	
59-62	Percent complete for current file	FixedPoint	
63-64	Number of fragments sent of current file	ShortInt	

field mask—Described in “Field Mask” on page 24.

OTAP state—Indicates the status of the OTAP process:

None	The OTAP process has been cancelled.
Initializing	OTAP handshakes are being performed in preparation for uploading files to the motes.
Upload	OTAP files are being uploaded to the motes.
Commit	Upload is now complete and motes are being reset in order to activate the new software. Cancelling an OTAP at this stage will result in motes running different software versions.
Completed	OTAP is completed.

getPath

Description

The *getPath* command returns the description for a given path.

Request Packet

The command takes the MAC address of the motes on either end of the path (mote A and mote B). The MAC address is packed as an array of bytes with the most significant byte first.

Table 73 getPath Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x37
3	Sequence	Byte	
4-11	Mote A MAC address	Byte[8]	
12-19	Mote B MAC address	Byte[8]	

mote A MAC address—The MAC address of the parent mote.

mote B MAC address—The MAC address of the child mote.

Response Packet

Table 74 getPath Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x38
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	

Table 74 `getPath` Response (Continued)

Message Byte	Description	Data Type	Value
9-12	Path ID	LongInt	
13-20	Mote A MAC address	Byte[8]	
21-28	Mote B MAC address	Byte[8]	
29-30	Number of links	ShortInt	

field mask—Described in “Field Mask” on page 24.

mote A MAC address—The MAC address of the parent mote.

mote B MAC address—The MAC address of the child mote.

number of links—Indicates how many links comprise the path.

getPathStatistics

Description

The `getPathStatistics` command returns a set of path statistics. Statistics may be requested for the current 15-minute period (on the quarter hour), a specific 15-minute period, a specific day, or for the lifetime of the mote (lifetime average).

Request Packet

Table 75 `getPathStatistics` Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x2B
3	Sequence	Byte	
4-7	Path ID	LongInt	
8	Statistics period	Byte	0x00 = Current 15-minute period 0x01 = Specific 15-minute 0x02 = Specific day 0x03 = Lifetime statistics
9	Period index	Byte	0x00 through 0x5F

statistics period, period index—Arguments used to specify the 15-minute period or the day for which statistics are requested:

- When specifying a 15-minute period, an index of 0 corresponds to the 15-minute period immediately preceding the current time. An index of 1 corresponds to the period before that one, and so on.
- When specifying a day, an index of 0 corresponds to the current day. An index of 1 corresponds to the day before that one, and so on.

For example, if you are using the index to specify a 15-minute period and it is currently 03:23, index 0 corresponds to the period from 03:00 to 03:15, and index 5 corresponds to the period from 01:45 to 02:00. The manager can store up to ninety-six 15-minute periods, so the index can range from 0 to 95.

If current or lifetime statistics are requested, the index is ignored.

Response Packet**Table 76** `getPathStatistics` Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x2C
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-12	Requested index	LongInt	
13-20	Start time of interval	DustTime	
21	Average RSSI mote B has to neighbor mote A	Byte	
22	Average RSSI mote A has to neighbor mote B	Byte	
23-26	Stability	FixedPoint	

The path statistics response contains a structure describing the path statistics for the requested time period.

field mask—Described in “Field Mask” on page 24.

requested index—The 15-minute period or day from the request packet.

start time of interval—The start time of the 15-minute period requested.

average RSSI mote B has to neighbor mote A—The average RSSI mote A (parent mote) has to neighbor mote B (child mote).

average RSSI mote A has to neighbor mote B—The average RSSI mote B (child mote) has to neighbor mote A (parent mote).

stability—The ratio of successful packet transmissions to the total number of packet transmissions on this path.

getPrevEvent

Description

The `getPrevEvent` command returns information about an event in the event log that occurred before the specified event ID number. This command can be used to iterate through all the events in the event log. Specifying event ID “0” returns the most recent event.

Request Packet**Table 77** getPrevEvent Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x2F
3	Sequence	Byte	
4-7	Event ID	LongInt	

Response Packet**Table 78** getPrevEvent Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x30
3	Sequence	Byte	
4	Response code	Byte	API_OK API_END_OF_LIST API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	Packet timestamp	DustTime	
13-16	Event ID	LongInt	
17	Event type	Byte	
18- <i>n</i>	Event payload		See "Events" on page 97

getSecurity**Description**

The *getSecurity* command returns the current network security parameters.

Request Packet**Table 79** getSecurity Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x39
3	Sequence	Byte	


Response Packet**Table 80** getSecurity Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x3A
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9	Security mode	Byte	0x00 = Accept ACL 0x01 = Accept common join key
10-42	Common join key	Byte[33]	
43	Accept HART devices only	Byte	0x00 = Accept all devices 0x01 = Accept only HART devices

field mask—Described in “Field Mask” on page 24.

security mode—Indicates which security mode is enabled. When “Accept ACL” is enabled, the manager allows only the motes on the access control list to join the network. When “Accept Common Join Key” is enabled, any mote that shares the manager’s network join key may join the network.

accept HART devices only—Indicates whether the network only accepts devices with a HART prefix in the MAC address. By default this field is set to accept all devices (0x00).

 **Note:** The join key is a symmetric encryption key that is shared between the manager the motes in its network. The join key is used by the manager and motes to encrypt and decrypt the join messages they exchange when the mote is attempting to join the network.

getSLA

Description

The *getSLA* command returns the network service level agreement (SLA) thresholds for minimum network reliability, maximum network latency, and minimum path stability. SLA thresholds define acceptable performance levels for the network. Alarms are generated whenever performance does not meet SLA thresholds.

Request Packet

Table 81 getSLA Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x35
3	Sequence	Byte	

Response Packet

Table 82 getSLA Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x36
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-12	Minimum network reliability	FixedPoint	
13-16	Maximum network latency (msec)	LongInt	
17-20	Minimum path stability	FixedPoint	

field mask—Described in “Field Mask” on page 24.

For other field descriptions, see “setSLA” on page 85.

getSystem

Description

The *getSystem* command returns a general system description, including the manager name, software version and current time value.

Request Packet

Table 83 getSystem Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x0F
3	Sequence	Byte	

Response Packet**Table 84** `getSystem` Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x10
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-25	Name	Char[17]	
26-42	Location	Char[17]	
43-44	Control TCP port	ShortInt	
45-46	Control SSL port	ShortInt	
47-48	Data TCP port	ShortInt	
49-50	Data SSL port	ShortInt	
51-52	Discovery UDP port	ShortInt	
53	SSL enabled	Byte	0x00 = False 0x01 = True
54-70	Software version	Char[17]	
71-87	Hardware model	Char[17]	
88-104	Hardware version	Char[17]	
105-121	Serial number	Char[17]	
122-129	Current time	DustTime	
130-137	Start time	DustTime	
138-139	CLI timeout	ShortInt	
140-155	Controller software version	Char[17]	

The system description contains system-level configuration parameters.

field mask—Described in “Field Mask” on page 24.

control TCP port, control SSL port, data TCP port, data SSL port, discovery UDP port—These values refer to the configuration for the XML API.

SSL enabled—If SSL enabled is true, the control TCP port and data TCP port are only accessible from the localhost, and stunnel is used to provide encrypted access to the control SSL port and data SSL port. If SSL enabled is false, the control TCP port and data TCP port provide plain text access for any client.

software version, hardware model and version, and serial number—Read-only values. Software version is the Dust Networks Controller ipackage version. The ipackage contains the Dust Networks Controller executable and other files such as Admin Toolset, setup scripts, and other Controller modules.

current time, start time—The current time and start time (timestamp of network manager startup).

CLI timeout—The time (seconds) after which an inactive command-line interface session is dropped (if CLI timeout is 0, there is no timeout).

controller software version—The version of the Dust Networks Controller’s executable.

getTime

Description

The *getTime* command returns a packet containing the current manager time and corresponding absolute slot number (ASN).

Request Packet

Table 85 getTime Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x0D
3	Sequence	Byte	

Response Packet

Table 86 getTime Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x0E
3	Sequence	Byte	
4	Response code	Byte	API_OK API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	System time	DustTime	
13-20	ASN	ASN	

system time, ASN—The Time structure contains system time (in seconds and microseconds) and absolute slot number (ASN). For information about *DustTime* and *ASN*, see “Data Types” on page 6.

pingMote

Description

The *pingMote* command sends a packet to a mote requesting a response. When the manager receives a reply from the mote, it will issue a *Ping Reply* event notification that includes this *callback ID* (see “Ping Reply” on page 107).

Request Packet

Table 87 pingMote Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x67
3	Sequence	Byte	
4-11	MAC address	Byte[8]	

Response Packet

If the packet is accepted, the manager returns the *callback ID* for this request. If the packet was not accepted by the manager, an error message is returned.

Table 88 pingMote Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x68
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_NETLAYER_FULL See Table 9 for response code descriptions.
5-8	Callback ID	LongInt	
9-16	MAC address	Byte[8]	

reset

Description

The reset command resets a mote, the network, or the system, or clears statistics or the event log. The command arguments include an object type and, depending on the type, an object ID. *The reset command is always sent as a best-effort packet.*

Request Packet**Table 89** reset Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x04
3	Sequence	Byte	
4	Reset type	Byte	0x01 = System 0x02 = Network 0x03 = Mote 0x04 = Statistics 0x05 = Event Log
5-12	MAC address	Byte[8]	

reset type—The following is a description of each type of reset:

- A system reset restarts the manager software processes, resets all statistics for the network, resets the manager’s wireless connection, and causes the network to reform.
- A network reset causes the manager’s wireless connection to reset; the entire network reforms as a result.
- A mote reset causes a specific mote to reboot and rejoin the network. A *Mote Reset* notification is issued when the manager sends the *reset* command to the mote (see “Mote Reset” on page 98). To verify that the mote has received the command and reset itself, use the *getMote* command to get the mote state.
- A statistics reset clears statistics on the network motes and the manager.
- An event log reset clears the event log.

MAC address—This parameter is used to reset a specific mote. The MAC address is packed as an array of bytes with the most significant byte first.

Response Packet**Table 90** reset Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x05
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_NETLAYER_FULL API_INVALID_COMMAND API_INVALID_ARGUMENT See Table 9 for response code descriptions.
5-12	MAC address	Byte[8]	

response code—The OK (0x00) response code indicates that the application layer has processed the command correctly.

MAC address—If a mote was reset, the response returns the MAC address of the mote. If a non-mote object (such as the network, system, or statistics) is reset, the MAC address field is empty.

sendRequest

Description

The *sendRequest* command sends a request packet to a specific mote.

Request Packet

Table 91 sendRequest Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x55
3	Sequence	Byte	
4-11	MAC address	Byte [8]	
12	Application domain	Byte	0x00 = Publish 0x01 = Event 0x02 = Maintenance 0x03 = Block transfer
13	Packet priority	Byte	0x00 = Low 0x01 = High 0x02 = Medium
14	isReliable	Byte	0x00 = Best effort 0x01 = Reliable
15	Serial data length	Byte	Length of the serial data in bytes
16...	Serial data	Byte []	Maximum of 78 bytes for WirelessHART packets.

MAC address—The MAC address is packed as an array of bytes with the most significant byte first. If the MAC address argument is set to 0xFF FF FF FF FF FF FF FF, the packet is broadcast to all motes in the network. Note that broadcast packets must be sent with *isReliable* set to “Best effort” (an error message is returned if broadcast packets are sent as “Reliable”).

application domain—Indicates the type of packet to be sent to the mote:

- Publish Data packets sent on a regular, periodic basis.
- Event Packets (such as alarms) that are triggered by an event.
- Maintenance A series of request/response packets used to manage the device.
- Block transfer A group of packets sent over a period of time.

isReliable—This argument allows the client to specify how the packet will be sent through the network. Using Dust’s reliable protocol ensures end-to-end acknowledgment of packet arrival at the mote. Best-effort still has link-level acknowledgments, but no end-to-end acknowledgements.

serial data length—The maximum data payload is 78 bytes. *Note that non-WirelessHART messages must prepend the 4-byte header, 0x00 00 FC12, leaving 74 usable bytes.* The full 78 bytes is usable for WirelessHART commands because the header is already included in the packet.

Response Packet

Table 92 sendRequest Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x56
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_CROSSREF API_VAL_PATTERN_HEX API_VAL_DATAPACKET API_NETLAYER_FULL API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Callback ID	LongInt	
9-16	MAC address	Byte[8]	

sendResponse

Description

The *sendResponse* command sends a response to a command or data packet that the mote sent to the client.

Request Packet

Table 93 sendResponse Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x57
3	Sequence	Byte	
4-11	MAC address	Byte [8]	
12	Application domain	Byte	0x00 = Publish 0x01 = Event 0x02 = Maintenance 0x03 = Block transfer
13	Packet priority	Byte	0x00 = Low 0x01 = High 0x02 = Medium

Table 93 sendResponse Request (Continued)

Message Byte	Description	Data Type	Value
14	isReliable	Byte	0x01 = True
15-18	Callback ID	LongInt	
19	Serial data length	Byte	Length of the serial data in bytes
20...	Serial data	Byte []	Maximum of 78 bytes for WirelessHART packets.

callback ID—The callback ID associated with the command or packet that was sent to the client.

For remaining field descriptions, including details about the serial data payload, see “sendRequest” on page 74.

Response Packet

Table 94 sendResponse Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x58
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_CROSSREF API_VAL_PATTERN_HEX API_VAL_DATAPACKET API_NETLAYER_FULL API_INVALID_COMMAND See Table 9 for response code descriptions.
5-12	MAC address	Byte[8]	

setAclDevice

Description

The *setAclDevice* command sets access control list (ACL) information for a specified mote. The ACL contains the motes that are allowed to join the network when the security mode is set to “Accept ACL.” For more information about security modes, see “setSecurity” on page 84.

Request Packet

Table 95 setAclDevice Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x4B
3	Sequence	Byte	
4-7	Field mask	Byte[4]	
8-15	MAC address	Byte[8]	
16-48	Join key	Char[33]	

field mask—Described in “Field Mask” on page 24.

MAC address—Packed as an array of bytes with the most significant byte first.

Response Packet

Table 96 setAclDevice Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x4C
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-16	MAC address	Byte[8]	
17-49	Join key	Char[33]	

response code—The OK (0x00) response code indicates that the application layer has processed the command correctly.


field mask—Described in “Field Mask” on page 24.

setChannelBlacklist

Description

The *setChannelBlacklist* command sets the channel blacklist, replacing any existing channel blacklist. Blacklisted channels are not used by the network. Although the network can operate on as few as five channels, it is recommended that the network be run on as many channels as possible for greater overall available bandwidth. Note that the number of blacklisted channels must be an odd number. An error is generated when the number of blacklisted channels is an even number.

When setting the channel blacklist, you are responsible for ensuring that the unblacklisted frequencies conform with local RF regulations. The sixteenth channel (frequency 2480) must be blacklisted for operation in the United States (as regulated by FCC) and Canada (as regulated by IC). The sixteenth channel is blacklisted by default, but can be unblacklisted using the *setChannelBlacklist* command (no error will be generated).

 **Note:** To return to the default channel blacklist, use the *setChannelBlacklist* command to set the channel blacklist to the default settings (all channels unblacklisted except the sixteenth channel, frequency 2480).

Request Packet**Table 97 setChannelBlacklist Request**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x45
3	Sequence	Byte	
4-7	Field mask	Byte[4]	
8-29	Frequency	ShortInt[11]	

field mask—Described in “Field Mask” on page 24.

frequency—Array of the blacklisted frequencies. The elements of the frequency array must be a value from 2405 to 2480 (inclusive, incrementing by 5). The array is always 11 elements long (at most 11 channels may be blacklisted). Unused elements must be set to 0 if there are fewer than 11 channels in the blacklist. Only an odd number of channels may be blacklisted.


Response Packet**Table 98 setChannelBlacklist Response**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x46
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-30	Frequency	ShortInt[11]	

response code—The OK (0x00) response code indicates that the application layer has processed the command correctly.

setMote**Description**

The *setMote* command configures the mote name and enables or disables data routing (one or both of these parameter values can be set) for a given mote. Note that the *name* field must be null terminated. The mote must be specified by either its *mote ID* or *MAC address*.

 **Note:** With the exception of the *mote ID* and *MAC address* fields, all other read-only fields in the packet payload must have their field mask bit set to (0) or an error message is generated and the command will be ignored.

Request Packet**Table 99 setMote Request**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x47
3	Sequence	Byte	
4-7	Field mask	Byte[4]	
8-9	Mote ID	ShortInt	Read-only
10-17	MAC address	Byte[8]	Read-only
18-34	Name	Char[17]	
35	Power source	Byte	Read-only 0x00 = Line 0x01 = Battery 0x02 = Recharge/power scavenging
36-37	Discharge current	ShortInt	Read-only
38-41	Discharge time	LongInt	Read-only
42-45	Recovery time	LongInt	Read-only
46	Enable routing	Byte	0x00 = Off 0x01 = On
47-63	Product name	Char[17]	Read-only
64-80	Hardware model	Char[17]	Read-only
81-97	Hardware revision	Char[17]	Read-only
98-114	Software revision	Char[17]	Read-only
115	isAccessPoint	Byte	Read-only 0x00 = Not an access point 0x01 = Access point
116-117	NumJoins	ShortInt	Read-only
118	State	Byte	See Table 47 Read-only
119	Mote state reason	Byte	Read-only 0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error
120-127	Join time	DustTime	Read-only
128-131	Voltage	FixedPoint	Read-only
132-133	Number of neighbors	ShortInt	Read-only
134-137	Allocated packet period	LongInt	Read-only
138-141	Allocated pipe packet period	LongInt	Read-only

Table 99 setMote Request (Continued)

Message Byte	Description	Data Type	Value
142	Pipe status	Byte	Read-only 0x00 = Off 0x01 = Action pending 0x02 = On (bidirectional) 0x03 = On (upstream) 0x04 = On (downstream)
143	Advertising status	Byte	Read-only 0x00 = On 0x01 = Off 0x02 = Action pending
144	Needs neighbor	Byte	Read-only 0x00 = False 0x01 = True

For field descriptions, see “getMote” on page 46.

Response Packet**Table 100 setMote Response**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x48
3	Sequence	Byte	
4	Response code	Byte	API_OK API_VAL_REQFIELD API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-10	Mote ID	ShortInt	
11-18	MAC address	Byte[8]	
19-35	Name	Char[17]	
36	Power source	Byte	0x00 = Line 0x01 = Battery 0x02 = Recharge/power scavenging
37-38	Discharge current	ShortInt	
39-42	Discharge time	LongInt	
43-46	Recovery time	LongInt	
47	Enable routing	Byte	0x00 = Off 0x01 = On
48-64	Product name	Char[17]	
65-81	Hardware model	Char[17]	

Table 100 setMote Response (Continued)

Message Byte	Description	Data Type	Value
82-98	Hardware revision	Char[17]	
99-115	Software revision	Char[17]	
116	isAccessPoint	Byte	0x00 = Not an access point 0x01 = Access point
117-118	NumJoins	ShortInt	
119	State	Byte	See Table 47
120	Mote state reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error
121-128	Join time	DustTime	
129-132	Voltage	FixedPoint	
133-134	Number of neighbors	ShortInt	
135-138	Allocated packet period	LongInt	
139-142	Allocated pipe packet period	LongInt	
143	Pipe status	Byte	0x00 = Off 0x01 = Action pending 0x02 = On (bidirectional) 0x03 = On (upstream) 0x04 = On (downstream)
144	Advertising status	Byte	0x00 = On 0x01 = Off 0x02 = Action pending
145	Needs neighbors	Byte	0x00 = False 0x01 = True

response code—The OK (0x00) response code indicates that the application layer has processed the command correctly.

For field descriptions, see “getMote” on page 46.

setNetwork

Description

The *setNetwork* command sets network configuration parameters, such as network ID, frame size, and optimization. One or more parameter values can be set at a time.

Request Packet**Table 101 setNetwork Request**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x41
3	Sequence	Byte	
4-7	Field mask	Byte[4]	
8-24	Network name	Char[17]	
25-26	Network ID	ShortInt	
27	Optimization	Byte	0x00 = Off 0x01 = On
28-35	Maintenance start time	DustTime	
36-43	Maintenance end time	DustTime	
44-45	Max notes	ShortInt	
46-47	Number of notes	ShortInt	Read-only
48	Access point power amplifier	Byte	0x00 = Off 0x01 = On
49	Clear channel access	Byte	0x00 = Off 0x01 = On
50-53	Requested base packet period	LongInt	
54-57	Minimum services packet period	LongInt	
58-61	Minimum pipe packet period	LongInt	
62	Bandwidth profile	Byte	See Table 54.
63-66	Manual upstream frame size	LongInt	
67-70	Manual downstream frame size	LongInt	
71-74	Manual advertising frame size	LongInt	
75-76	Network queue size	ShortInt	Read-only
77-78	User queue size	ShortInt	Read-only

For field descriptions, see “getNetwork” on page 52.

Response Packet

Table 102 setNetwork Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x42
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_UNCHANGEABLE API_VAL_MINMAX API_VAL_TIMEBOUNDARY API_VAL_STARTEND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-25	Network name	Char[17]	
26-27	Network ID	ShortInt	
28	Optimization	Byte	0x00 = Off 0x01 = On
29-36	Maintenance start time	DustTime	
37-44	Maintenance end time	DustTime	
45-46	Max motes	ShortInt	
47-50	Number of motes	LongInt	
51	Access point power amplifier	Byte	0x00 = Off 0x01 = On
52	Clear channel access	Byte	0x00 = Off 0x01 = On
53-56	Requested base packet period	LongInt	
57-58	Minimum services packet period	LongInt	
59-62	Minimum pipe packet period	LongInt	
63	Bandwidth profile	Byte	See Table 54.
64-67	Manual upstream frame size	LongInt	
68-71	Manual downstream frame size	LongInt	
72-75	Manual advertising frame size	LongInt	
76-77	Network queue size	ShortInt	Read-only
78-79	User queue size	ShortInt	Read-only

response code—The OK (0x00) response code indicates that the application layer has processed the command correctly.

For field descriptions, see “getNetwork” on page 52.

setSecurity

Description

The *setSecurity* command sets the network security parameters.

Request Packet

Table 103 setSecurity Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x49
3	Sequence	Byte	
4-7	Field mask	Byte[4]	
8	Security mode	Byte	0x00 = Accept ACL 0x01 = Accept common join key
9-41	Common join key	Byte[33]	
42	Accept HART devices only	Byte	0x00 = Accept all devices 0x01 = Accept only HART devices

For field descriptions, see “getSecurity” on page 67.

Response Packet

Table 104 setSecurity Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x4A
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9	Security mode	Byte	0x00 = Accept ACL 0x01 = Accept common join key
10-42	Common join key	Byte[33]	
43	Accept HART devices only	Byte	0x00 = Accept all devices 0x01 = Accept only HART devices

For field descriptions, see “getSecurity” on page 67.

setSLA

Description

The *setSLA* command sets the network service level agreement (SLA) thresholds for minimum network reliability, maximum network latency, and minimum path stability. SLA thresholds define acceptable performance levels for the network. Alarms are generated whenever performance does not meet SLA thresholds.

Request Packet

Table 105 setSLA Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x43
3	Sequence	Byte	
4-7	Field mask	Byte[4]	
8-11	Minimum network reliability	FixedPoint	
12-15	Maximum network latency (msec)	LongInt	
16-19	Minimum path stability	FixedPoint	

field mask—Described in “Field Mask” on page 24.

minimum network reliability—Network reliability is the ratio of the number of packets received at the manager to the number of packets expected at the manager.

maximum network latency—Network latency is the packet latency, measured in milliseconds.

minimum path stability—Path stability is the ratio of the number of acknowledgments received to the number of acknowledgements expected.

Response Packet

Table 106 setSLA Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x44
3	Sequence	Byte	
4	Response code	Byte	API_OK API_END_OF_LIST API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-12	Minimum network reliability	FixedPoint	
13-16	Maximum network latency (msec)	LongInt	
17-20	Minimum path stability	FixedPoint	

response code—OK (0x00) response code indicates that the application layer has processed the command correctly.

field mask—Described in “Field Mask” on page 24.

setSystem

Description

The *setSystem* command sets system configuration parameters, such as data and control ports. One or more different parameter values can be set at a time. Note that the *name* field must be null terminated.

Request Packet

Table 107 setSystem Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x3F
3	Sequence	Byte	
4-7	Field mask	Byte[4]	
8-24	Name	Char[17]	
25-41	Location	Char[17]	
42-43	Control TCP port	ShortInt	
44-45	Control SSL port	ShortInt	
46-47	Data TCP port	ShortInt	
48-49	Data SSL port	ShortInt	
50-51	Discovery UDP port	ShortInt	
52	SSL enabled	Byte	0x00 = False 0x01 = True
53-69	Software version	Char[17]	Read-only
70-86	Hardware model	Char[17]	Read-only
87-103	Hardware version	Char[17]	Read-only
104-120	Serial number	Char[17]	Read-only
121-128	Current time	DustTime	Read-only
129-136	Start time	DustTime	Read-only
137-138	CLI timeout	ShortInt	
139-155	Controller software version	Char[17]	Read-only

For field descriptions, see “getSystem” on page 69.

Response Packet

The response packet returns the current system settings.

Table 108 setSystem Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x40
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_CROSSREF API_VAL_UNCHANGEABLE API_INVALID_COMMAND See Table 9 for response code descriptions.
5-8	Field mask	Byte[4]	
9-25	Name	Char[17]	
26-42	Location	Char[17]	
43-44	Control TCP port	ShortInt	
45-46	Control SSL port	ShortInt	
47-48	Data TCP port	ShortInt	
49-50	Data SSL port	ShortInt	
51-52	Discovery UDP port	ShortInt	
53	SSL enabled	Byte	0x00 = False 0x01 = True
54-70	Software version	Char[17]	
71-87	Hardware model	Char[17]	
88-104	Hardware version	Char[17]	
105-121	Serial number	Char[17]	
122-129	Current time	DustTime	
130-137	Start time	DustTime	
138-139	CLI timeout	ShortInt	
140-156	Controller software version	Char[17]	

The OK response code indicates that the application layer has processed the command correctly.

For field descriptions, see “getSystem” on page 69.

startOtap

Description

The *startOtap* command triggers the manager to start an OTAP (over-the-air programming) session.

Request Packet

Table 109 startOtap Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x61
3	Sequence	Byte	
4	Number of retries	Byte	

number of retries—Specifies the number of times the manager should try uploading the software to the mote if the previous attempt fails.

Response Packet

Table 110 startOtap Response

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x62
3	Sequence	Byte	
4	Response code	Byte	API_OK API_OBJECT_NOT_FOUND API_VAL_STATE API_INVALID_COMMAND See Table 9 for response code descriptions.

The OK (0x00) response code indicates that the application layer has processed the command correctly.

subscribe

Description

The *subscribe* command is used to set up a connection to the notification channel, over which a client program may receive data, network alarms, or network events.

Request Packet

Each subscription request overwrites the previous one. For example, if a client is subscribed to data samples and then decides he wants alarms as well, he should send a subscribe command with both the data sample and alarm flags set. To clear all subscriptions, the client should send a subscribe command with the reliable filter and best effort filters set to zero.

Table 111 Subscribe Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x0A
3	Sequence	Byte	
4-7	Reliable filter	Byte[4]	See Table 112
8-11	Best-effort filter	Byte[4]	See Table 112

The notification filters (*reliable filter* and *best-effort filter*) are bit masks of flags indicating the types of notifications that the client wants to receive. If a *subscribe* command comes with a notification type set to both reliable and best-effort, an error will be returned and the command will be ignored.

reliable filter—The reliable filter parameter indicates the type of notifications that the client wishes to receive using the serial reliable protocol (see “Reliable Communication” for details).

best-effort filter—The best-effort filter parameter indicates the type of notifications that the client wishes to receive without the reliable protocol.

Table 112 Reliable and Best-effort Bit Masks

Notification	Bitmask
Serial data	0x0000 0001
Alarms	0x0000 0004
System events	0x0000 0008
Topology events	0x0000 0010
System errors	0x0000 0020
Log messages	0x0000 0040
CLI responses	0x0000 0080

Response Packet**Table 113 Subscribe Response**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x0B
3	Sequence	Byte	
4	Response code	Byte	API_OK API_INVALID_ARGUMENT See Table 9 for response code descriptions.

The OK (0x00) response code indicates that the application layer has processed the command correctly. Any other response code indicates an error.

Remarks

After the *subscribe* response, asynchronous packets will be written by the manager to the serial interface. These packets will contain the data notifications or events requested by the client. If the notification requires an acknowledgement, any subsequent reliable notification packets will be queued while waiting for the acknowledgement.

When a new session is established between the client and the manager, there are no subscriptions.

Notifications

Notifications are asynchronous messages from the SmartMesh manager to the client, and are used to deliver data packets, alarms, and events. For example, when a data packet is received from a mote in the wireless network, the SmartMesh manager will send a notification message to its clients with the contents of the serial packet. Alarm notifications are sent when network performance metrics reach an unacceptable level. Event notifications capture occurrences such as the network being reset, or a new mote joining the network. Table 114 summarizes the types of notifications and the following sections describe each in detail.

Table 114 Notifications

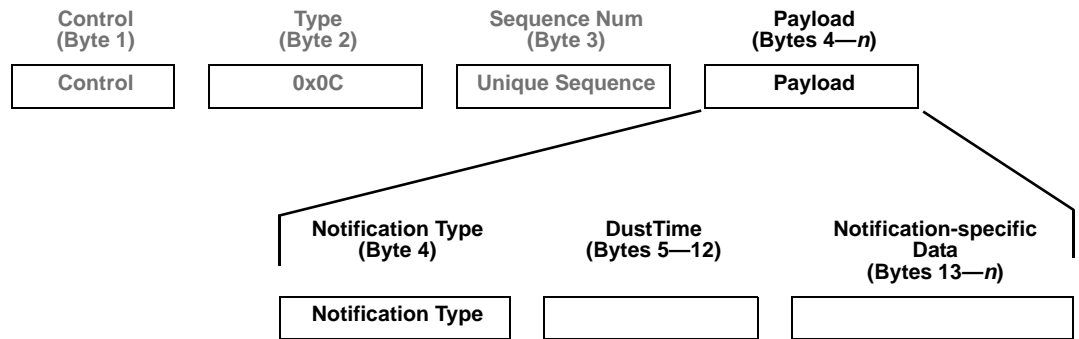
Notification	Description	Notification Type
Serial Data Notification	Contains serial data sent from a mote in the wireless network	0x00
Alarm Notifications		0x01
Alarm Close	An alarm condition has been corrected.	
Mote Down	A mote is no longer responding to messages from the manager.	
Low Reliability	The network reliability dropped below the preset SLA threshold (see “setSLA” on page 85).	
High Latency	The network latency increased above the preset SLA threshold (see “setSLA” on page 85).	
Low Stability	The network stability dropped below the preset SLA threshold (see “setSLA” on page 85).	
Maximum Number of Motes Reached	The maximum number of motes (as specified using “setNetwork” on page 81) was reached.	
System Event Notifications	See “Events” on page 97.	0x02
Network Event Notifications	See “Events” on page 97.	0x03
System Error Notifications	Information used for network troubleshooting.	0x04
Log Notifications	Information used for network troubleshooting.	0x05
CLI Notifications	The user-readable response payload for a CLI command previously sent.	0x06

A client uses the *subscribe* command to select the types of notifications it wants to receive. A client application may subscribe to one or more event notifications. After receiving a *subscribe* request (and sending the confirmation response), the manager will send notification packets to the client that match the requested event types. Since at most one packet can be outstanding in the link layer, the notification packets will be interleaved with acknowledgements and command responses.

All notification packets have the packet type 0x0C (notification type). The notification structure is described below for each type of notification.

If a client receives a notification type that it does not understand, it must acknowledge the notification if appropriate, but should then skip to the end of the notification and continue.

Notification Packet Format



Note that in best-effort communications, the *Sequence Number* field is unused and set to zero. For details about best-effort communication and reliable communication, see “Session Packet Processing.”

The *DustTime* field is the time the packet was generated and is described in “Data Types” on page 6.

Notification Details

The following sections provide a byte-level description of each notification packet. Refer to the previous section for information about Sequence Number usage.

Serial Data Notification Packet

The *serial data notification* packet contains serial data sent from a mote (identified by its MAC address) in the wireless network. It is a variable length array of binary data. The length of serial data is determined based on the length of the notification packet.

Table 115 Send Data Notification Packet

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x00
5–12	Timestamp	DustTime	
13–20	MAC address	Bytes [8]	
21	isReliable	Byte	0x00 = False 0x01 = True
22	isRequest	Byte	0x00 = False 0x01 = True
23	isBroadcast	Byte	0x00 = False 0x01 = True
24-27	callbackId	LongInt	
28–n	Serial data	Byte []	Maximum of 94 bytes.

isReliable—Indicates how the packet was sent through the network (see “Reliable Communication” on page 16).

isRequest—This field is true for requests and false for responses.

isBroadcast—This field is always “false.”

callback ID—The callback ID associated with this packet, if the packet is a reliable packet.

serial data—The serial data payload can be a maximum of 94 bytes.

Alarm Notification Packet

SmartMesh manager can send alarm packets when a performance metric of a mote, path, or the entire network has reached an unacceptable level. Each alarm has alarm-specific data.

The *Alarm Close* event indicates that an alarm event has been resolved. All other alarms indicate that an alarm has been opened.

Table 116 Alarm Types

Value	Description
0x00	Alarm Close
0x01	Mote Down
0x02	Low Reliability
0x03	High Latency
0x04	Low Stability
0x05	Reserved
0x06	Maximum Number of Motes Reached

Alarm Close

This packet is sent when a specific alarm is closed.

Table 117 Alarm Close Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x01
5-12	Packet timestamp	DustTime	
13-16	Alarm ID	LongInt	
17	Alarm type	Byte	0x00
18-21	Alarm open ID	LongInt	
22	Alarm open type	Byte	Alarm types are described in Table 116.

Mote Down

A mote is considered *down* when it is no longer responding to messages from the manager.

Table 118 Mote Down Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x01
5-12	Packet timestamp	DustTime	
13-16	Alarm ID	LongInt	
17	Alarm type	Byte	0x01
18-25	MAC address	Byte[8]	

Low Reliability

Data reliability is defined as the percentage of expected data packets that were actually received. One hundred percent reliability means that every expected data packet was received. A low reliability alarm is generated when the network average goes below the minimum level set in the service level agreement (see “setSLA” on page 85). By default, this limit is 99%.

Table 119 Low Reliability Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x01
5-12	Packet timestamp	DustTime	
13-16	Alarm ID	LongInt	
17	Alarm type	Byte	0x02

High Latency

Data latency is the average time (in milliseconds) required for a data packet to travel from the originating mote to the manager. Latency varies across the network. A high latency alarm is generated when the average latency for the entire network exceeds the limit set in the service level agreement (see “setSLA” on page 85). By default, this limit is 12.5 seconds.

Table 120 High Latency Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x01
5-12	Packet timestamp	DustTime	
13-16	Alarm ID	LongInt	
17	Alarm type	Byte	0x03

Low Stability

Path stability is a measure of mote-to-mote transmissions. It is the percentage of data packets that are successfully acknowledged at each hop. If a transmitting mote does not receive an acknowledgement it will resend the packet on an alternate path. Due to the unique benefits of mesh routing, data reliability can be 100% even with very low path stability. A *Low Stability* alarm is generated when the network-wide average of path stability falls below the minimum limit set in the service level agreement (see “setSLA” on page 85). By default, this limit is 50%.

Table 121 Low Stability Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x01
5-12	Packet timestamp	DustTime	
13-16	Alarm ID	LongInt	
17	Alarm type	Byte	0x04

Maximum Number of Motes Reached

SmartMesh manager keeps a limit on the number of motes that may join the network. This limit can be configured using the “setNetwork” command described on page 81. A *Max Number of Motes Reached* notification is generated when the network has reached its configured limit (by default, the limit is 50 motes).

Table 122 Maximum Number of Motes Reached Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x01
5-12	Packet timestamp	DustTime	
13-16	Alarm ID	LongInt	
17	Alarm type	Byte	0x06

System Event Notification Packet

System events describe system-level (often user requested) changes.

Table 123 System Event Notification Packet

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x02
5-12	Packet timestamp	DustTime	
13-n	Event payload		See “Events” on page 97.

Network Event Notification Packet

Network events describe various events related to changes in the network topology. Unlike an alarm, which opens when a condition occurs and closes when the condition is no longer true, a network event is sent only once when the condition changes.

Table 124 Network Event Notification Packet

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x03
5-12	Packet timestamp	DustTime	
13-n	Event payload		See "Events" on page 97.

System Error Notification Packet

System Error notifications may be requested by Dust Networks Support to help identify a problem during network troubleshooting.

Table 125 System Error Notification Packet

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x04
5-12	Packet timestamp	DustTime	
13	Severity	Byte	
14-n	Error message	Char []	Maximum 1011 bytes.

Log Notification Packet

Log notifications may be requested by Dust Networks Support to help identify a problem during network troubleshooting.

Table 126 Log Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x05
5-12	Packet timestamp	DustTime	
13	Severity	Byte	
17-n	Log message	Char []	Maximum 1011 bytes.

CLI Notification Packet

Command line interface (CLI) notifications contain the user-readable response payload for a CLI command that was previously sent. Each *CLI* notification contains one line of the CLI output. For information about the CLI command, see “cli” on page 32.

Table 127 CLI Notification

Message Byte	Description	Data Type	Value
1	Control	Byte	
2	Type	Byte	0x0C
3	Sequence	Byte	
4	Notification type	Byte	0x06
5-12	Packet timestamp	DustTime	
13	End of response	Byte	0x00 = False 0x01 = True
14-n	CLI message	Char []	Maximum 1011 bytes.

Events

This section describes the packet payloads (bytes 13-*n*) of system and network events. The payload information is used in system and network event notifications (see Tables 123 and 124 on page 95) and in the response packet of the *getPrevEvent* command (see “getPrevEvent” on page 66). The appropriate header information is prepended to the packet payload.

Table 128 Events

Event	Description	Event Type
System Events		
Mote Reset	A mote was reset.	0x00
Mote Deleted	A mote was deleted from the network.	0x01
Network Reset	The wireless network was reset.	0x02
Manager Reset	The SmartMesh manager software process was reset.	0x03
Mote Decommissioned	A mote is being decommissioned from the system.	0x04
System Bootup	The system started up.	0x05
System Connection	A user connected to the system.	0x06
System Disconnection	A user disconnected from the system.	0x07
Reset Statistics	A user reset all statistics.	0x08
Configuration Change	A user changed the system configuration.	0x09
Command Finished	An exchangeJoinKey, exchangeNetworkId, or exchangeSessionKey command has completed execution.	0x22

Table 128 Events (Continued)

Event	Description	Event Type
Network Events		
Mote Join	A mote joined the network.	0x10
Mote Live	A new mote was configured.	0x11
Mote Unknown	A mote is no longer communicating in the network.	0x12
Mote Disconnect	A mote in the network is ready to be physically removed (traffic has been re-routed to make this mote a leaf node).	0x13
Mote Join Failure	A mote failed to join the network.	0x1E
Mote Invalid MIC	An invalid packet was received from a mote. An invalid message integrity check (MIC) may indicate a security problem.	0x1F
Pipe On	A pipe was activated for a mote.	0x0F
Pipe Off	A pipe was deactivated for a mote.	0x14
Frame Change	A frame's state changed.	0x1C
Packet Sent	A data packet was injected into the network.	0x1D
Service Denied	The manager failed to allocate a requested service.	0x20
Ping Reply	A reply was received from a mote ping.	0x21
Internal Reset	System reset the network.	0x1B
Path Create	A connection (path) was created between two motes in the network.	0x0A
Path Delete	A connection (path) was removed between two motes in the network.	0x0B
Path Activate	Communication was restored on a path (the two motes have a link in an active frame).	0x0C
Path Deactivate	A network path was deactivated. The path will continue to exist until it is deleted. The path can be reused if it is activated again.	0x0D
Path Alert	The manager received an alarm about this path.	0x0E

System Events

System events describe system-level (often user-requested) changes.

Mote Reset

This notification is sent when the manager sends a *reset* command to a mote. It is also sent for the access point mote when the manager starts up. Note that sending a *reset* command to a mote does not guarantee that the mote has received the command and reset itself. Use the *getMote* command to determine the current mote state.

Table 129 Mote Reset Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-34	User name	Char[17]	
35-42	MAC address	Byte[8]	

Mote Deleted

This notification is sent when a mote is deleted from the network.

Table 130 Mote Deleted Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-34	User name	Char[17]	
35-42	MAC address	Byte[8]	

Network Reset

This notification is sent when the network is about to be reset.

Table 131 Network Reset Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-34	User name	Char[17]	

Manager Reset

This notification is sent when the SmartMesh manager software process is reset.

Table 132 Manager Reset Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-34	User name	Char[17]	

Mote Decommissioned

This notification is sent when a mote is decommissioned. Decommissioning a mote prepares the mote for removal from the network. The manager re-routes traffic to make the mote a leaf node (a node with no other motes reporting to it). When the mote is ready to be removed from the network, a *Mote Disconnect* event is generated (see “Mote Disconnect” on page 104).

Table 133 Mote Decommissioned Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-34	User name	Char[17]	
35-42	MAC address	Byte[8]	

System Bootup

This notification is sent when the system starts up.

Table 134 System Boot Up Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128

System Connection

This notification is sent when a user connects to the system.

Table 135 System Connection Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-33	User name	Char[17]	
34-49	User IP address	Char[16]	
50-57	Channel	Byte[8]	0x00 = CLI 0x01 = Configuration 0x02 = Notification

System Disconnection

This notification is sent when a user disconnects from the system.

Table 136 System Disconnection Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-33	User name	Char[17]	
34-49	User IP address	Char[16]	
50-57	Channel	Byte[8]	0x00 = CLI 0x01 = Configuration 0x02 = Notification

Reset Statistics

This notification is sent when a user resets all statistics.

Table 137 Reset Statistics Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128

Configuration Change

This notification is sent when a user changes the configuration of a system element, such as a mote, path, network ID/key, session key, or SLA settings. This notification is most useful when multiple clients are simultaneously connected to the manager.

Table 138 Configuration Change Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-34	User name	Char[17]	
35	Object type	Byte	0x00 = Mote 0x01 = Path 0x02 = Network 0x03 = Network ID 0x04 = Network key 0x05 = Join key 0x06 = Session key 0x07 = SLA 0x08 = System 0x09 = User 0x0A = Blacklist 0x0B = Security 0x0C = ACL
36-59	Object ID	Char[24]	

Command Finished

This notification is sent when an *exchangeJoinKey*, *exchangeNetworkId*, or *exchangeSessionKey* command is completed.

Table 139 Command Finished Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Callback ID	LongInt	
22-29	MAC address	Byte[8]	
30	Object type	Byte	0x03 = Network ID 0x04 = Network key 0x05 = Join key 0x06 = Session key
31	Result code	LongInt	0x00

Network Events

Network events describe various events related to changes in the network topology. Unlike an alarm, which opens when a condition occurs and closes when the condition is no longer true, a network event is sent only once when the condition changes

Mote Join

This notification is sent when a mote joins the network.

Table 140 Mote Join Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	
26-106	User data	Char[81]	

Mote Live

This notification is sent when the mote is configured.

Table 141 Mote Live Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	
26	Reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error

Mote Unknown

This notification is sent once when the mote changes to the Unknown state, which may occur for one of two reasons:

- A mote is considered unknown when it becomes unreachable through the network. In this case the manager generates a one-time *Mote Unknown* network event as well as a *Mote Down* alarm (see “Mote Down” on page 93).
- A mote may also be unknown at network startup if it has not yet joined. This only occurs if the mote has previously joined the network and is not deleted from the mote list.

Table 142 Mote Unknown Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	
26	Reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error

Path Create

This notification is sent when a new connection (path) is created between two motes (mote A and mote B).

Table 143 Path Create Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Path ID	LongInt	
22-29	MAC address of mote A	Byte[8]	
30-37	MAC address of mote B	Byte[8]	

Path Delete

This notification is sent when a connection (path) is deleted between two motes (mote A and mote B).

Table 144 Path Delete Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Path ID	LongInt	
22-29	MAC address of mote A	Byte[8]	
30-37	MAC address of mote B	Byte[8]	

Path Activate

This notification is sent when communication is restored on a path. The two motes on either end of the path (mote A and mote B) have a link in an active frame.

Table 145 Path Activate Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Path ID	LongInt	
22-29	MAC address of mote A	Byte[8]	
30-37	MAC address of mote B	Byte[8]	

Path Deactivate

This notification is sent when communication is lost on a path. The two motes on either end of the path (mote A and mote B) have no links in an active frame.

Table 146 Path Deactivate Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Path ID	LongInt	
22-29	MAC address of mote A	Byte[8]	
30-37	MAC address of mote B	Byte[8]	

Path Alert

This notification is sent when the manager receives an alarm about a path.

Table 147 Path Alert Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Path ID	LongInt	
22-29	MAC address of mote A	Byte[8]	
30-37	MAC address of mote B	Byte[8]	

Pipe On

This notification is sent when a pipe is activated between the manager and a mote (identified by its MAC address).

Table 148 Pipe On Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	

Mote Disconnect

This notification is sent when a mote has entered the “Disconnect” state. The mote is ready to be physically removed from the network (traffic has been re-routed to make this mote a leaf node).

Table 149 Mote Disconnect Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	
26	Reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error

Pipe Off

This notification is sent when a pipe is deactivated between the manager and a mote (identified by its MAC address).

Table 150 Pipe Off Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	

Internal Reset

This notification is sent when the system resets the network.

Table 151 Internal Reset Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128

Frame Change

This notification is sent when a frame's state has changed.

Table 152 Frame Change Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18	Frame ID	Byte	
19	Frame state	Byte	0x00 = Inactive 0x01 = Active 0x02 = Activating 0x03 = Deactivating

Packet Sent

This notification is sent when a *sendRequest* or *sendResponse* command is issued, causing a data packet (identified by its callback ID) to be injected into the network. The MAC address is the destination address of the data packet.

For information about the *sendRequest* command, see “sendRequest” on page 74.

For information about the *sendResponse* command, see “sendResponse” on page 75.

Table 153 Packet Sent Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Callback ID	LongInt	
22-29	MAC address	Byte[8]	

Mote Join Failure

This notification is sent when a mote (identified by its MAC address) fails to join the network.

Table 154 Mote Join Failure Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	
26	Reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error

Mote Invalid MIC

This notification is sent when an invalid packet is received from a mote (identified by its MAC address). An invalid message integrity check (MIC) may indicate a network security problem.

Table 155 Mote Invalid MIC Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-25	MAC address	Byte[8]	
26	Reason	Byte	0x00 = None 0x01 = Max motes reached 0x02 = Unreachable 0x03 = Not connected 0x04 = Configuration error

Service Denied

This notification is sent when the manager fails to allocate a service requested by a mote.

Table 156 Service Denied Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-19	Service ID	ShortInt	
20-27	MAC address of mote requesting service	Byte[8]	
28-35	MAC address of peer device for service	Byte[8]	
36	Application domain	Byte	0x00 = Publish 0x01 = Event 0x02 = Maintenance 0x03 = Block transfer
37	Service requested as source	Byte	0x00 = False 0x01 = True
38	Service requested as sink	Byte	0x00 = False 0x01 = True
39	Service requested as intermittent	Byte	0x00 = False 0x01 = True
40-43	Service period/latency	LongInt	

peer device—The peer device for the service is the destination for the service. For example, the virtual gateway could be the destination for service from a mote.

application domain—This field indicates the type of packet to be sent to the mote:

- Publish Data packets sent on a regular, periodic basis.
- Event Packets (such as alarms) that are triggered by an event.
- Maintenance A series of request/response packets used to manage the device.
- Block transfer A group of packets sent over a period of time.

service requested as source—This field indicates whether the mote was to be the source of data generated.

service requested as sink—This field indicates whether the mote was to be the receiver of data.

service requested as intermittent—This field indicates whether the service was requested to support regular reporting or intermittent traffic.

service period/latency—This field indicates the length of the service period for non-intermittent service, or the period between service packets for intermittent service.

Ping Reply

This notification is sent when a reply is received from a mote ping. *Callback ID* is the callback ID associated with the ping.

Table 157 Ping Reply Event Payload

Message Byte	Description	Data Type	Value
13-16	Event ID	LongInt	
17	Event type	Byte	See Table 128
18-21	Callback ID	LongInt	
22-29	MAC address of mote pinged	Byte[8]	
30-33	Latency (msec)	LongInt	
34-35	Temperature (°C) reported by mote	ShortInt	
36-37	Voltage (V) reported by mote	ShortInt	
38	Upstream hop count for ping reply	Byte	



Serial Port Login

This appendix describes the four logins available on the serial port—manager serial API, PPP, Linux shell, and manager CLI. For more information about how the serial port functions, see “Serial Port Process Switch” on page 2.

Manager Serial API Login

Logging in to the serial port as “serial” provides access to the manager serial API. Upon login, the serial API process is started on the serial port expecting a serial API client connection. If no client connection is received within 60 seconds, the serial API process terminates and the serial port returns to the login prompt. Once the client connects, there is no timeout for automatic disconnection. To disconnect, the client should issue a disconnect command to close down the serial API and return to the login prompt.

Use the following username and password to log in as a serial user:

```
Username: serial
Password: serial
```

PPP Login

To establish a PPP connection, configure your PPP client to detect the login prompt from the serial port and log in as the “ppp” user. An example of this configuration is shown below. Upon login, the manager starts a PPP server on the serial port expecting a PPP connection. If no PPP connection is established within 120 seconds (2 minutes), the PPP process terminates and the serial port returns to the login prompt. Once the connection is established, manager's PPP server sends periodic keepalive messages to verify that the client is still connected. If an insufficient number of keepalives are acknowledged, the PPP process automatically terminates. For PPP interface specifications, refer to the *SmartMesh 510 PM2511 Datasheet*.

The PPP login can be used for operations such as:

- *Sending XML API commands to the manager*—For instructions on logging in to manager's control channel, see the *SmartMesh IA-510 XML API Guide*.
- *Transferring files to or from the manager*—For example, you can use a process such as Secure Copy (SCP) or Secure FTP (SFTP) to transfer a software update package to manager, or transfer log files from manager to the client.

- *Troubleshooting the network through manager's CLI*—For example, you can use a client such as SSH or PuTTY to open a connection to the manager CLI. For information about manager's CLI commands, see the *SmartMesh IA-510 CLI Commands Guide*.

Logging in to the Manager Using a Linux Client

If the client is a Linux system, use the following username and password to access PPP over the serial port:

Username: ppp

Password: ppp

You can use the following scripts to automate the PPP login to a Linux client. The PPP options file defines a chat script that automates the login procedure. The chat script sends a return character to get the first login prompt, looks for the string ogin: and sends ppp as the username, and look for the string assword: and send ppp as the password.

/etc/ppp/options.ttyS1:

```
passive
115200
192.168.101.10:192.168.101.11
connect '/usr/sbin/chat -v -t 30 -f /etc/ppp/pm1200-ppp.chat'
```

/etc/ppp/pm1200-ppp.chat:

```
TIMEOUT 30
'' \n
ogin:--ogin:    ppp
assword:       ppp
```

Logging in to the Manager Using a Windows Client

If the client is a Windows system (which requires a handshake to start up), use the following username and password to access to PPP over the serial port:

Username: pppw

Password: pppw

For information on how to configure a Windows XP client for a PPP connection, see Appendix C.

Linux Shell Login

The manager allows login to a Linux shell (using a terminal emulator, such as HyperTerminal) for enabling the manager APIs, reading log files, and setting common configuration options such as IP address changes.

Use the following username and password to access the Linux login prompt.

Username: `dust`

Password: `dust`

To disconnect and return to the login prompt, type `logout`. If there is no activity for 3600 seconds (1 hour), the connection is automatically terminated.

Table 158 describes key Linux commands that are supported on the manager. Some commands provide usage information if you type “`-help`” as the first argument. For example, “`sudo set-time -help`”.


 **Note:** When configuring a static IP address (`sudo ifswitch-to-static` command), you need to power cycle the manager with an active Ethernet connection in order for the IP configuration to display the correct Ethernet configuration.

Table 158 Key Linux Commands Supported on the Manager

Command	Description
<code>sudo ifswitch-to-static</code>	Changes the network interface to use a specified static IP address.
<code>sudo ifswitch-to-dhcp</code>	Changes the network interface to use a IP address that is dynamically assigned by the LAN.
<code>sudo set-time</code>	Sets manager's internal clock. Use this command only when configuring manager; do not change manager's internal clock after motes have joined and the mesh network is running.
<code>create-network-snapshot</code>	Creates a snapshot of the current manager and network state.
<code>sftp</code>	Standard Linux command for performing a secure file transfer.
<code>scp</code>	Standard Linux command for performing a secure file transfer.

Manager CLI Login

You can log on to the manager CLI using a terminal emulator such as HyperTerminal to connect to the serial port (use VT100 for the terminal emulation). The manager CLI commands allow you to view information about the network and perform administrative tasks, such as updating the network ID or security keys.

Use the following username and password to log in to the manager CLI.

Username: dustcli

Password: dustcli

The CLI connection will terminate automatically if there is no activity within the timeout period. The timeout period can be set using the manager XML or serial API, or the manager CLI commands. For information about the manager APIs, refer to the *SmartMesh IA-510 Manager XML API Guide* or the *SmartMesh IA-510 Manager Serial API Guide*. For information about the manager CLI commands, refer to the *SmartMesh IA-510 CLI Commands Guide*. The default timeout period is 120 minutes (2 hours).

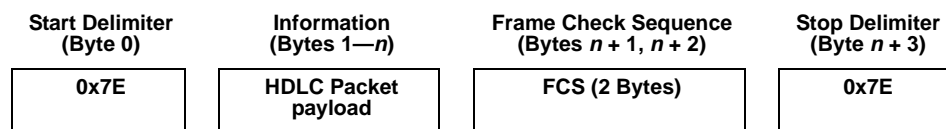
To disconnect and return to the login prompt, type logout.

HDLC Packet Format

This appendix describes the HDLC packet framing used by the IA-510 manager serial protocol, and provides examples of HDLC packet processing.

HDLC Packet

The following diagram summarizes the packet format:



Start/Stop Delimiters

The protocol uses 0x7E for packet delineation. Every packet must start and end with this flag.

Information

The *Information* field is IA-510 manager protocol-specific and is defined in the following sections. Note that it does *not* contain HDLC Control and Address fields that are described in RFC1662. These fields are not supported, and should not be used.

Frame Check Sequence

The frame check sequence field (FCS) is used to check validity of packets received. The field is calculated over all bytes of the Information portion of each packet, excluding any start/stop bits that may be added for asynchronous transmission. This specifically does not include the start/stop delimiter or the FCS field itself. It is recommended that the field be calculated using the algorithm specified in RFC1662.

Octet Stuffing

The IA-510 manager serial protocol uses octet stuffing to escape start/stop delimiter (0x7E) and Control Escape (0x7D) bytes that may be contained in the Information or FCS fields. Async-Control-Character-Map (ACCM) mechanism, as defined in RFC 1662, is *not* used, so all other bytes values can be sent un-escaped.

After FCS computation, the transmitter examines the entire frame between the start and stop delimiter. Each 0x7E and 0x7D (excluding the flags) is then replaced by a two-byte sequence consisting of the Control Escape (0x7D) followed by the original byte exclusive-or'd with 0x20.

To summarize:

- 0x7e in payload is transmitted as 0x7d, 0x5e
- 0x7d in payload is transmitted as 0x7d, 0x5d
- On reception, each Control Escape (0x7D) is removed and the following byte is exclusive-or'd with 0x20, unless it is 0x7E (which aborts the frame)

HDLC Packet Processing Examples

Example 1: Constructing an HDLC packet to send

This example demonstrates how to construct an HDLC packet. (All values are in hexadecimal.)

Step 1 Assume the following 10-byte HDLC packet payload:

HDLC Packet Payload
03 07 02 00 00 00 00 03 00 7D

Step 2 Calculate FCS:

- Calculate the FCS using FCS-16 algorithm (RFC 1662) on the hexadecimal sequence 03 07 02 00 00 00 00 03 00 7D. The FCS (including 1's complement) is B2 9A.
- Append FCS to payload, FCS is sent least significant byte first (RFC 1662):

HDLC Packet Payload	FCS
03 07 02 00 00 00 00 03 00 7D	9A B2

Step 3 Perform byte stuffing.

To perform byte stuffing, check the HDLC Packet Payload and FCS for instances of “7D” or “7E” and replace as follows:

7D=> 7D 5D

7E=> 7D 5E

Note that the additional control bytes do not count against the message payload limit, as defined in “Packet Communication and Format” on page 5.

HDLC Packet Payload (stuffed)	FCS (stuffed)
03 07 02 00 00 00 00 03 00 7D 5D	9A B2

Step 4 Add start and stop delimiters.

Enclose the above in start/stop flags (RFC 1662).

Note that some products may require an additional 7E start byte for high speed operation. Please refer to product datasheets.

Start Byte	HDLC Packet Payload (stuffed)	FCS (stuffed)	Stop Byte
7E	03 07 02 00 00 00 00 03 00 7D 5D	9A B2	7E

Or simply, the hexadecimal sequence:

7E 03 07 02 00 00 00 00 03 00 7D 5D 9A B2 7E

Example 2: Deconstructing an HDLC packet received

To understand how to deconstruct an HDLC packet, assume that the following HDLC Packet was received. (All values are in hexadecimal.)

Start Byte	HDLC Packet Payload (stuffed)	FCS (stuffed)	Stop Byte
7E	04 09 01 00 01 00 00 00 00 00 00 7D 5E C3	C9 D0	7E

Step 1 (HDLC layer) strip off delimiters.

HDLC Packet Payload (stuffed)	FCS (stuffed)
04 09 01 00 01 00 00 00 00 00 00 7D 5E C3	C9 D0

Step 2 Remove byte stuffing.

To remove byte stuffing, check for instances of “7D 5D” or “7D 5E” and replace as follows:

7D 5D=> 7D

7D 5E=> 7E

HDLC Packet Payload	FCS
04 09 01 00 01 00 00 00 00 00 00 7E C3	C9 D0

Step 3 Confirm FCS.

Calculate the checksum for the HDLC payload.

HDLC Packet Payload
04 09 01 00 01 00 00 00 00 00 00 7E C3

Confirm that the FCS matches the FCS sent with the packet. Because the packet encodes FCS least significant byte first, in this example the calculated FCS should match “D0 C9”.

Configuring a Windows XP PC for PPP



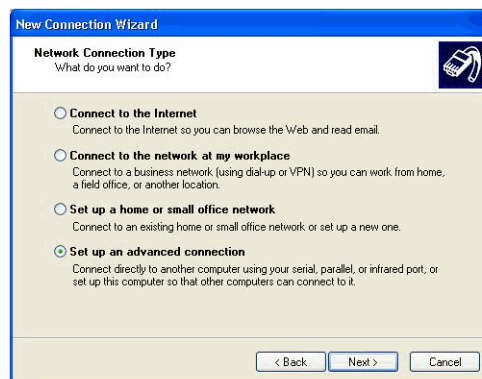
Use the following procedure to configure a Windows XP PC for a PPP connection to the manager.

To configure a Windows XP based PC for PPP:

- 1 Using a DB9 non-null modem cable, connect the serial port on the client PC to the serial port on the manager.
- 2 Click **Start=>Control Panel**. Then click **Network Connections**.
- 3 Click **Network Connection Wizard** icon.



- 4 Click **Next**.
- 5 Click **Set up an advanced connection** and click **Next**.



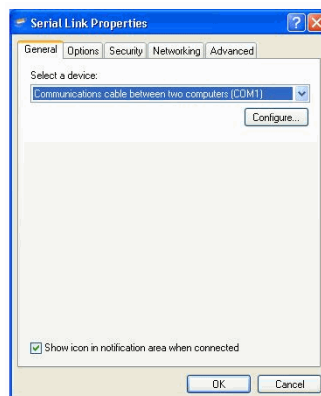
- 6 Click **Connect directly to another computer**, and then click **Next**.



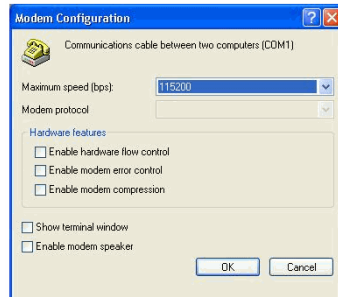
- 7 Click **Guest**, and then click **Next**.
- 8 For the computer name, type **Serial Link** and click **Next**.
- 9 Select the serial port you will use (typically COM1) and click **Next**.
- 10 Select the option to add the connection icon to your desktop, and then click **Finish**.
The Connect dialog box should display. If it does not, double-click the connection icon on your desktop.



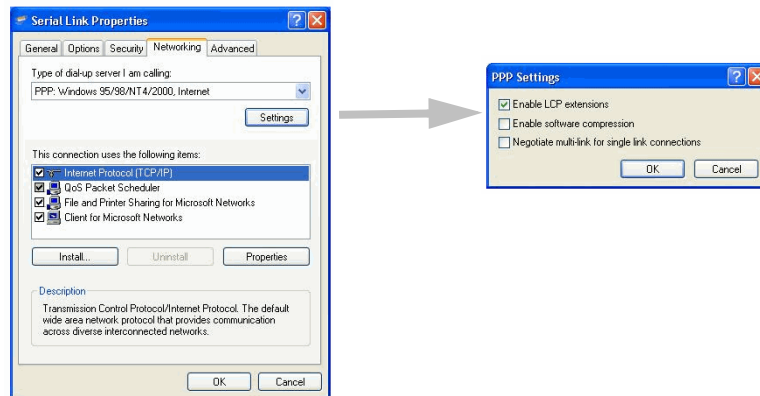
- 11 Click **Properties**. Make sure the serial port is selected, and then click **Configure**.



- 12 Change the maximum speed to **115200**, and select the **Enable hardware flow control** option. Click **OK**.



- 13 Click the **Networking** tab. Click **Settings** and select the **Enable Software Compression** option. Click **OK**.




- 14 Open a terminal emulator (such as HyperTerminal) connection to the serial port.
 15 Login using “**pppw**” as the login username and password.
 16 Close the terminal emulator connection.
 17 In the Connect window, click **Connect** to connect to the manager via PPP.



By default, the IP address of the manager over the PPP connection is 192.168.101.10, and the client PC's PPP IP address is 192.168.101.11. You can now use an XML API application to connect to the manager at its IP address. See the *SmartMesh IA-510 Manager XML API Guide* for details on how to use the XML RPC API.

Updating Software

This appendix provides procedures for updating the manager and mote software. Always check the readme file that accompanies the software update package for information about software compatibility and processes that may need to be performed prior to the software update.

 **Note:** If you are updating software on both the mote and manager, complete the mote software update before initiating the manager software update.

To update the mote software, use these procedures:

- Transfer the software update package to the manager.
- Initiate the mote software update.

To update the manager software, use these procedures:

- Transfer the software update package to the manager.
- Initiate the manager software update.

Transferring a Software Update Package to the Manager

You can use a PPP connection or a non-IP based serial connection to the manager to transfer the software update package (ipackage file) to the default directory on the manager.

To transfer the update package using a PPP connection:

- ❶ Open a terminal session to the serial port.
- ❷ Initiate a PPP connection to the manager.
For instructions, see “PPP Login” in Appendix A.
- ❸ Transfer the ipackage file using a secure transfer program, such as SFTP or SCP.

To transfer the update package using a non-IP based serial connection:

- ❶ Open a terminal session to the serial port using a program, such as HyperTerminal, that supports the z-modem protocol.
- ❷ Log in using as a Linux user:
Username:dust
Password:dust
- ❸ Transfer the ipackage file using the z-modem protocol.

Initiating a Mote Software Update Package

After transferring the software update package to the manager (see the previous procedure), follow these steps to install the mote software in the manager's OTAP directory and then initiate a mote software update by executing the start OTAP command via the XML API, manager serial API, or manager CLI.

The start OTAP command updates software for network motes and the access point (AP) mote on the manager. For command details, refer to the following guides:

- *SmartMesh IA-510 Manager XML API Guide*
- *SmartMesh IA-510 Manager Serial API Guide*
- *SmartMesh IA-510 Manager CLI Commands Guide*

To install the mote update package in the OTAP directory:

- 1 Open a terminal session to the serial port.
- 2 Log in as the Linux user (if not already logged in):
Username:dust
Password:dust
- 3 Go to the directory into which you copied the file when you transferred it to the manager.
- 4 Issue the command: `sudo ipkg install <filename>`
For <filename> specify the ipackage file containing the software update. This command extracts the mote and access point (AP) file from the ipackage and places these files in the manager's /root/otap directory.
! **Important:** If you are reinstalling an ipackage file that you installed previously, you must first remove the ipackage file. Use the following command sequence:
`sudo ipkg remove <filename>`
`sudo ipkg install <filename>`
- 5 Initiate the mote software update by executing the start OTAP command via the XML API, manager serial API or manager CLI. Refer to the following procedures for detailed instructions.

To use the manager CLI to initiate an OTAP:

- 1 Log in as a manager CLI user:
Username:dustcli
Password:dustcli
- 2 Issue the command: `exec startOtap`
For command details, refer to the *SmartMesh IA-510 Manager CLI Commands Guide*.

- ③ Wait until the OTAP process is completed. You can use the `get otapstatus` command to check the OTAP status.
- ④ When the OTAP process is completed, issue the `exec reset network` command to reset the network.

To use the manager serial API to initiate an OTAP:

- ① Open a terminal session to the port.
- ② Initiate a connection to the serial API.
For instructions, see “Manager Serial API Login” in Appendix A.
- ③ Send a `startOtap` command request.
For command details, see the *SmartMesh IA-510 Manager Serial API Guide*.
- ④ Wait until the OTAP process is completed. You can use the `getOtapStatus` command to check the OTAP status.
- ⑤ When the OTAP process is completed, issue the `exec reset network` command to reset the network.

To use the manager XML API to initiate an OTAP:

- ① Open a terminal session to the serial port.
- ② Initiate a PPP connection to the manager.
For instructions, see “PPP Login” in Appendix A.
- ③ Log in to the control channel on manager.
- ④ Issue the `startOtap` command to begin the OTAP process.
- ⑤ Wait until the OTAP process is completed. You can use the `getConfig` command to check the OTAP status.
For command details, refer to the *SmartMesh IA-510 XML API Guide*.
- ⑥ When the OTAP process is completed, issue the `exec reset network` command to reset the network.

Initiating a Manager Software Update

If you are updating software on both the mote and manager, complete the mote software update (see the previous procedure) before initiating the manager software update.

Follow these steps to install the software update on the manager after transferring the update package to the manager, as described in “Transferring a Software Update Package to the Manager” earlier in this appendix.


To initiate a manager software update:

- ❶ Open a terminal session to the manager serial port.
- ❷ Log in to the serial port using the Linux login:
Username:dust
Password:dust
- ❸ Go to the directory into which you copied the file when you transferred it to the manager.
- ❹ Issue the command: `sudo ipkg install <filename>`
For <filename> specify the ipackage file containing the software update.
 - ! **Important:** If you are reinstalling an ipackage file that you installed previously, you must first remove the ipackage file. Use the following command sequence:
`sudo ipkg remove <filename>`
`sudo ipkg install <filename>`
 - ! **Caution:** During installation, you will be asked if you want to overwrite the configuration file. Do not overwrite this file unless you want to reset the manager's network ID and join key to the factory default settings. If you overwrite the configuration file, you will need to re-configure these settings on the manager in order for the network to re-form.
- ❺ Once the software is installed, issue the command: `sudo reboot`
The manager restarts and the new software is activated.

Adjusting Manager Time

adjustTime

The *adjustTime* command changes the time on the manager to the specified reference time. The manager makes the time adjustment gradually over time so that manager operation can continue uninterrupted. The time required to make the adjustment depends on the magnitude of the adjustment (the difference between the reference time and the current manager time). For example, for each second of adjustment the manager gradually updates the clock over 2,000 seconds of real-world time. If the *adjustTime* command is re-issued during this period, the manager will return an error (API_NOTUNIQCMD). A maximum adjustment of 600 seconds can be made in a single *adjustTime* request. Corrections over 600 seconds can be made by issuing multiple requests. However, it is recommended that you monitor time periodically to avoid the need for large adjustments.

 **Note:** The *adjustTime* command requires that the manager's NTP server is enabled and configured with the NTP server address at 127.127.28.0. This may be done using the Linux command `sudo set-time`. Because `sudo set-time` requires a manager reset, it is recommended that you do this prior to network operation, for example, during system installation. Refer to Appendix A for Linux login information.

Request Packet

The reference time should be the time at which this API command is sent to the manager. The manager compensates for the time it takes to make the adjustment. The reference time is expressed as the number of microseconds since Epoch (midnight January 1, 1970), separated into the seconds and microseconds components. Note that the byte order for both reference time fields is reversed. For example, for a value of 100, enter 0x64 0x00 0x00 0x00.

Table 159 adjustTime Request

Message Byte	Description	Data Type	Value
1	Control	Byte	0x02
2	Type	Byte	0x70
3	Sequence	Byte	
4-7	Reference time (seconds)	LongInt	
8-11	Reference time (microseconds)	LongInt	

Response Packet**Table 160 adjustTime Response**

Message Byte	Description	Data Type	Value
1	Control	Byte	0x03
2	Type	Byte	0x71
3	Sequence	Byte	
4	Response code	Byte	API_NOTUNIQCMD API_VAL_STATE API_VAL_REQFIELD API_INVALID_COMMAND API_OBJECT_NOT_FOUND API_UNIDENTIFIED See Table 9 for response code descriptions.

Index

A

absolute slot number. *See* ASN

ACK packet, 17
control field, 17
empty ACKs, 19
payload field, 17
sequence number field, 17
type field, 17

ACL device
deleteACLDevice, 35
getACLDevice, 43
getNextACLDevice, 55
setACLDevice, 76

activateAdv, 29, 30

activateFastPipe, 29, 31

adjustTime, 27

advertising, 30

alarm notification packet, 92

alarm types, 93
alarm close, 93
high latency, 94
low reliability, 94
low stability, 94
max number of motes reached, 95
mote down, 93

alarms, 56

ASN, 6, 71

B

best-effort packet, 15

broadcast packet, 74

byte ordering, 7

C

cancelOtap, 29, 32

channel blacklist
getChannelBlacklist, 44
setChannelBlacklist, 77

cli, 29, 32

command summary, 27

communication failure, 19

control field, 7

ACK packet, 17
reliable packet, 16

D

data types, 6

deactivateFastPipe, 29, 33

decommissionDevice, 29, 34

deleteACLDevice, 28, 35

deleteMote, 28, 35

delimiters, 113

disconnect, 27, 36

DustTime, 6, 71

E

error handling, 19

exchangeJoinKey, 28, 37

exchangeMoteJoinKey, 28, 38

exchangeNetworkId, 28, 39

exchangeNetworkKey, 28, 41

exchangeSessionKey, 28, 42

F

FCS

field mask, 24

frame check sequence *See* FCS

full-duplex communication, 17

G

getACLDevice, 28, 43

getChannelBlacklist, 28, 44

getLatency, 29, 45

getMote, 28, 46

getMotePaths, 60

getMoteStatistics, 29, 49

getNetStatistics, 29, 51

getNetwork, 28, 52

getNextACLDevice, 28, 55

- getNextAlarm, 29, 56
- getNextMote, 28, 57
- getNextOtapMote, 29
- getOtapFile, 29
- getOtapMote, 29
- getOtapStatus, 29, 63
- getPath, 28
- getPathStatistics, 29, 65
- getPrevEvent, 29, 66
- getSecurity, 28, 67
- getSLA, 29, 69
- getSystem, 28, 69
- getTime, 28, 71

H

- HDLC packet format, 113
- HELLO packet, 8, 13, 17
- HELLO_RESPONSE packet, 13

I

- information field, 113

J

- join key, 37, 38

M

- manager reset, detecting, 10
- manager time, 71
- MGR_HELLO packet, 12
- mote
 - activating advertising, 30
 - decommissioning, 34
 - deleteMote, 35
 - getMote, 46
 - getMotePaths, 60
 - getMoteStatistics, 49
 - getNextMote, 57
 - join key, 38
 - latency estimates, 45
 - mote states, 49
 - resetting, 72
 - setMote, 78
- mote states, 49

N

- network
 - getNetStatistics, 51
 - getNetwork, 52

- resetting, 72
- setNetwork, 81

- network event packet, 96, 102

- network event types
 - frame change, 105
 - internal reset, 105
 - mote invalid MIC, 106
 - mote join, 96, 102
 - mote join failure, 105
 - mote live, 102
 - mote unknown, 102
 - packet sent, 105
 - path activate, 103
 - path alert, 104
 - path create, 103
 - path deactivate, 103
 - path delete, 103
 - ping reply, 107
 - pipe off, 104
 - pipe on, 104
 - service denied, 106

- network ID, 39

- network key, 41

- network security mode, 67

- notification packet, 91

- notifications, 90

O

- octet stuffing, 113

OTAP

- cancelOtap, 32
- getOtapStatus, 63
- startOtap, 88

P

- packet contents
 - control field, 7
 - payload field, 8
 - sequence number field, 8
 - type field, 8

- packet field descriptions, 7

packet format

- HDLC packet, 113
- HELLO packet, 13
- HELLO_RESPONSE packet, 13
- MGR_HELLO packet, 12
- notification packet, 91

path

- getMotePaths, 60
- getPathStatistics, 65

- payload field, 8

- ACK packet, 17
- reliable packet, 16

- pingMote, 29, 72

- pipe, 31, 33

R

- reliable packet, 16
- reset, 29, 72
 - detecting manager reset, 10
- response codes, 25

S

- security, 67
- security mode, setting, 84
- sendPacket, 74
- sendRequest, 27, 74
- sendResponse, 27, 75
- sequence number field, 8
 - ACK packet, 17
 - reliable packet, 16
- sequence numbers, 17
- serial API server, shutting down, 36
- serial data notification packet, 92
- serial port
 - login, 109
 - process switch, 2
- serial session, establishing, 8
- session key, 42
- setACLDevice, 28, 76
- setChannelBlacklist, 28, 77
- setMote, 28, 78
- setNetwork, 28, 81
- setSecurity, 28, 84
- setSLA, 29, 85
- setSystem, 28, 86
- SLA
 - getSLA, 69
 - setSLA, 85
- SmartMesh-enabled Network, 1
- startOtap, 29, 88
- state machine
 - client, 10
 - manager, 11
- statistics
 - getMoteStatistics, 49
 - getNetStatistics, 51
 - getPathStatistics, 65
- subscribe, 27, 88
- system
 - getSystem, 69
 - resetting, 72
 - setSystem, 86
- system event packet, 95
- system event types
 - system decommission of mote, 99
 - system delete of mote, 99
 - system reset of manager, 99
 - system reset of mote, 98

- system reset of network, 99

T

- type field, 8
 - ACK packet, 17
 - reliable packet, 16

U

- updating software, 121

