

Dust Networks

SmartMesh® IA-510

LM2650 Manager Serial API Guide



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. 2010. All Rights Reserved

Document Number: 040-0083 rev 3 IA-510 (L) LM2650 Manager Serial API Guide

Last Revised: August 20, 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	v
Conventions Used	v
Revision History	v

1 Introduction

SmartMesh-enabled Networks	1
LM2650 Manager APIs	2
Serial API	2
Bootloader API	2

2 Using the Serial API

Packet Communication and Format	3
Byte Ordering and Data Types	4
Packet Field Descriptions	4
Control	4
Type	5
Sequence Number	5
Payload	5
Establishing a Session	6
Establishing a Serial API Session	6
Establishing a New Session after OEM Controller Reboot	7
Detecting a Manager Reset	8
OEM Controller State Machine	8
Manager State Machine	9
Session Establishment Commands	10
MGR_HELLO	11
HELLO	11
HELLO_RESPONSE	12
Session Packet Processing	13
Session Packet Processing Guidelines	13

Example of Failed Communication	14
End-to-end Wireless Communication	14
Reliable Wireless Communication	14
Best-effort Wireless Communication	15
Access Control List	15
Network Bandwidth Control	16
Response Codes	16
Commands	17
clearStatistics	18
delAclEntry	19
exchangeNetworkId	20
getLog	20
getMote	21
Mote States	23
getMoteStatistics	23
getNetStatistics	25
getNetwork	26
getNextAclEntry	27
getNextPathStatistics	27
getSystem	29
getTime	29
pingMote	30
radioTestRx	31
radioTestStats	32
radioTestTx	32
reset	34
sendRequest	35
setAclEntry	36
setNetwork	37
subscribe	38
Notifications	39
Notification Packet Format	39
Notification Details	40
Serial Data Notification Packet	40
Event Notification Packet	41
Log Notification Packet	41
Events	42
Forward Compatibility	44

A Bootloader API

Overview	45
Software Update File Format	47
Establishing a Bootloader Session	47
How to Use Bootloader Commands	48
Bootloader Commands	48
prepare	49
write	49
read	50
verify	51
boot	52
reset	52
shutDown	53

B HDLC Packet Format

HDLC Packet	55
Start and Stop Delimiters.....	55
Information.....	55
Frame Check Sequence.....	55
Octet Stuffing	56
HDLC Packet Processing Examples	56

Index


About This Guide

This guide provides instructions for using the SmartMesh® IA-510 LM2650 manager serial protocol to communicate with the manager over its asynchronous serial port.

Related Documents

The following documents are available for SmartMesh IA-510 LM2650 products:

- *IA-510 LM2650 Manager Serial API Guide* (this guide)
- *IA-510 LM2650 Datasheet*
- *IA-510 DLM2650 Manager Guide*


 **Note:** For information on configuring and connecting to the serial API, refer to the product datasheet for the embedded LM2650 manager (*IA-510 LM2650 Datasheet*) or the user guide for the packaged manager (*IA-510 DLM2650 Manager Guide*).


Conventions Used

The following conventions are used in this guide:

- `Computer type` indicates information that you enter, such as specifying a URL.
- *Italic type* is used to indicate a command or field, and introduce new terms.

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

 **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
040-0083 rev 1	5/3/2010	Final publication.
040-0083 rev 2	6/29/2010	
040-0083 rev 3	8/20/2010	

Introduction

This chapter introduces the SmartMesh IA-510 LM2650 manager 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 transmission.
- A manager that configures the network and motes, receives data packets from the motes and streams data to the OEM controller, and receives data and other request packets from the OEM controller and streams them to the motes.
- The OEM controller that provides the manager with configuration instructions, receives mote and network data from the manager in serial format, and sends data and requests to the motes through the manager.

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

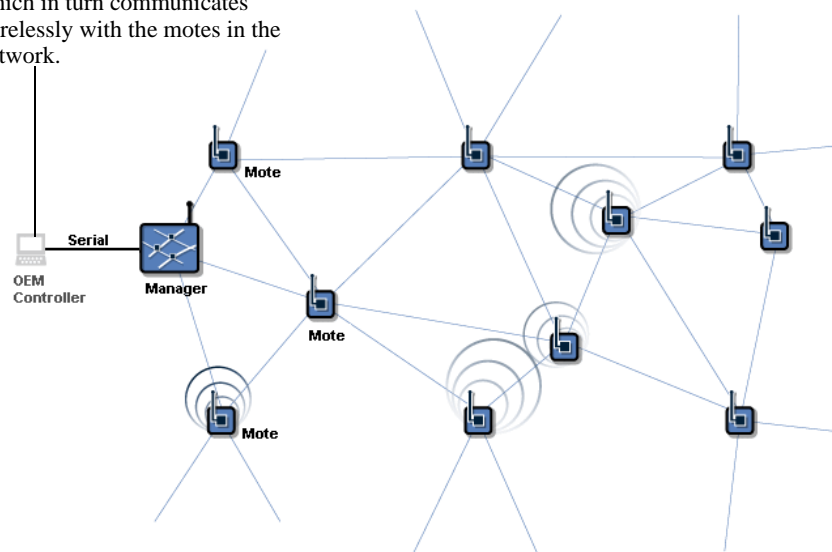


Figure 1 SmartMesh-enabled Network

LM2650 Manager APIs

The SmartMesh IA-510 LM2650 manager can function in either *bootloader mode* or *operational mode*.

- **Bootloader mode**—Allows OEM controllers to use the bootloader API commands to update the manager or access point software. The SmartMesh wireless network is not operational when manager is in bootloader mode.
- **Operational mode**—Allows OEM controllers to use the serial API commands to manage and configure a SmartMesh wireless network and send serial data to and receive serial data from the wireless notes.

The bootloader API and serial API are packet-based serial protocols that allow the OEM controller to communicate with the manager over its asynchronous serial port.

When the LM2650 manager is first powered on, it starts in bootloader mode and waits for the OEM controller to establish a bootloader session. If a session is not established with the bootloader within the timeout period (12 seconds), the manager switches to operational mode and waits for the OEM controller to establish a session with the manager application (see Figure 2). The process of establishing a session is described in Chapter 2, along with detailed information about packet format, packet processing, and data types.

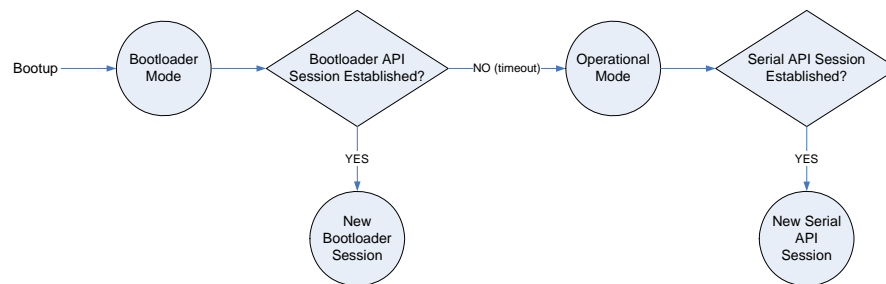


Figure 2 LM2650 Bootup Process

Serial API

The serial API mode is used for managing and configuring a SmartMesh wireless network and for sending serial data to and receiving data from wireless notes. The serial API provides two types of serial communication:

- **Commands**—These are two-way request and response messages. Commands are used to send serial data to wireless notes, configure and manage the wireless network, and subscribe to the notification channel.
- **Notifications**—These are asynchronous messages sent from the manager to the OEM controller. Notifications can contain serial data from wireless notes and information about network events and alarms.

Chapter 2 provides detailed information about serial API commands and notifications.

Bootloader API

The bootloader API mode allows the OEM controller to access manager's flash in order to update the manager or access point software. The bootloader API commands are described in Appendix A.

Using the Serial API

This chapter describes the LM2650 serial protocol and how to establish a session between the OEM and the manager. It also provides detailed information about serial commands, notifications, and events.

Packet Communication and Format

The LM2650 uses a packet-based serial protocol over an asynchronous serial port. The protocol runs at 115 kbps, 8 bits, no parity, 1 stop bit. RTS/CTS hardware handshaking is used, as described in the product datasheet.

The serial protocol for the LM2650 manager uses byte-oriented HDLC framing for all packets, as outlined in RFC 1662 (see <http://rfc.net/rfc1662.html>). The following diagram summarizes the HDLC packet format. Appendix B provides a description of HDLC fields and examples of HDLC packet processing.

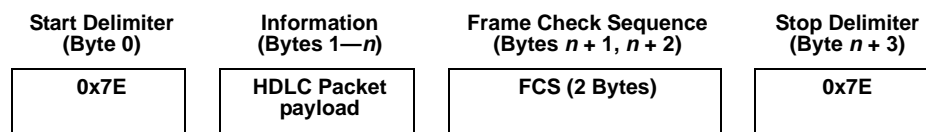


Figure 3 HDLC Packet Format

All packets in the LM2650 manager protocol contain the following fields in the *information* section of HDLC packet. The HDLC packet payload can be a maximum of 128 bytes.

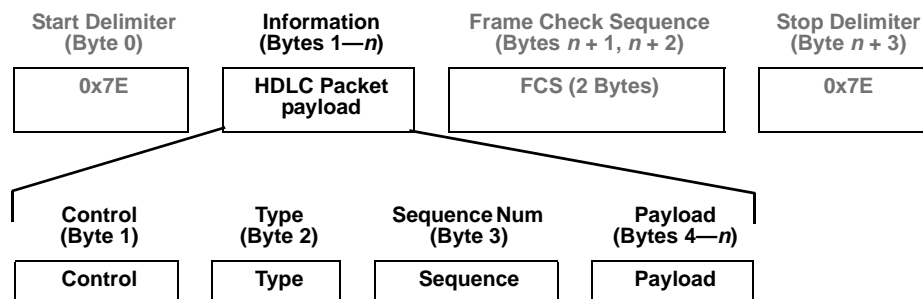


Figure 4 LM2650 Protocol Packet Format

Byte Ordering and Data Types

All values sent over the LM2650 manager serial protocol should be sent in big-endian order. The data appears on the wire MSB-first. This document uses the following data types to describe the command packets.

Table 1 Data Types

Data Type	Definition
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.
Char[n]	Fixed length character array. Strings specified as fixed length character arrays (for example, char[16]) are terminated with at least one null character. Values shorter than the full size are padded with null characters.
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 ($65536 \times 0.5 = 32768$).

Packet Field Descriptions

This section describes the *control*, *type*, *sequence number*, and *payload* fields contained in the LM2650 protocol (shown in Figure 4).

Control

The *control* field is a mandatory bitmap field contained in the beginning of every packet. It describes whether the packet is a request packet or a response packet, and whether the packet is acknowledged or unacknowledged. All packets are acknowledged except the *HELLO* packets that are exchanged during session establishment.

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

Table 2 Control Field

Field	Definition
Bit 0	0 = Request packet 1 = Response packet
Bit 1	0 = Unacknowledged (only used for packets exchanged during session establishment) 1 = Acknowledged (used for all packets except those exchanged during session establishment)
Bits 2–7	Reserved, set to 0

Control Field in Request Packets

For request packets, the first bit of the *control* field equals 0 and the second bit of the *control* field equals 1, resulting in a *control* byte value of 0x02.

Control Field in Response Packets

For response packets, the first bit of the *control* field equals 1 and the second bit of the *control* field equals 1, resulting in a *control* byte value of 0x03.

Type

Type is a mandatory field indicating the type of packet that follows. Each command, has a unique type number. For *type* values, refer to Table 3 for session establishment packets, Table 11 for command packets, and Table 56 for notification packets.

Sequence Number

The *sequence number* field is used for all communication packets. 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 as follows:

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

Once the session is established (the *HELLO* packet exchange is completed), the first request packet in the session should use the next *sequence number* following the number received during the *HELLO* packet exchange. In response packets, this field contains the *sequence number* of packet being acknowledged.

Payload

The *Payload* field is an optional field, the presence and contents of which are determined by the individual command.

Payload Field in Request Packets

The maximum size of the message payload is 90 bytes. For information on payload compatibility with future Dust Networks products contact Dust Networks Support.

Payload Field in Response Packets

The *payload* field in the response packet contains a response code and any data associated with the response (see “Response Codes” on page 16).

Establishing a Session

This section describes how an OEM controller may establish and maintain a session with the manager application.

The *HELLO* packet exchange starts a new session between the OEM controller and the manager. The *mode* field in the packet indicates whether the packet is from the bootloader or the manager application (see Figure 5).

The handshake is used to clear previous settings, agree on protocol version, and establish sequence numbers for future communication.

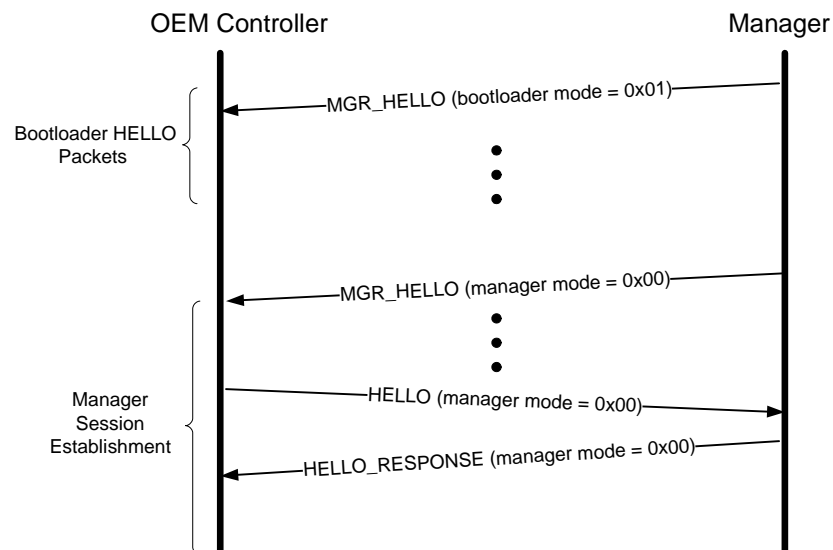



Figure 5 Establishing a Session

Establishing a Serial API Session

When the manager is powered on or reset, the bootloader launches and sends four *MGR_HELLO* packets at three-second intervals. If a session is not established with the bootloader, the manager application takes over and begins sending periodic *MGR_HELLO* packets. The OEM controller should use the following steps to establish a new session with the manager.

 **Note:** For information on establishing a bootloader session, see Appendix A.

To establish a new session with the manager:

- ❶ Generate periodic *HELLO* packets (once per second is the recommended interval) with the *mode* set for serial API. For each new packet generated, increment the *client sequence number* (start with *client sequence number* 0).
- ❷ If a valid *HELLO_RESPONSE* packet is received (*response* code is *RC_OK* and the *client sequence number* and *mode* 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 (*response* code is not *RC_OK* or the *mode* does not match the *HELLO* packet), the OEM controller can choose to adjust the *HELLO* message parameters (for example, the *version* field) and resend the *HELLO* packet.
- ④ If no *HELLO_RESPONSE* packet is received, the OEM controller should continue resending *HELLO* until a response is received.
- ⑤ Any other packet (including *MGR_HELLO*) received in response to *HELLO* packet must be dropped.

Upon completing the *HELLO* packet exchange, both the OEM controller and the manager should do the following.

- 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 OEM Controller Reboot

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

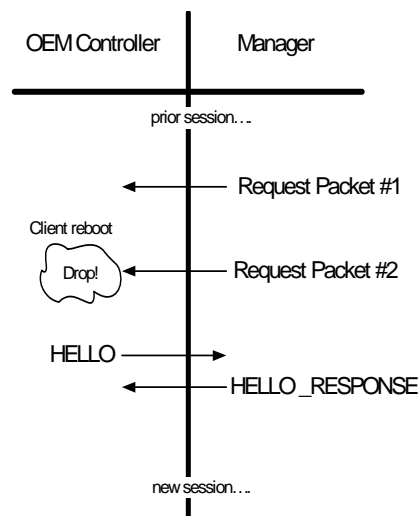


Figure 6 Establishing New Session Following OEM Controller Reboot

Detecting a Manager Reset

If the manager terminates its session due to communication errors or reset, it will revert back to the starting state. Upon receiving a *MGR_HELLO* packet, the OEM controller should assume the previous session has been terminated, and attempt to reestablish connection using the *HELLO* and *HELLO_RESPONSE* exchange.

Figure 7 shows how a session is reestablished following a manager reset.

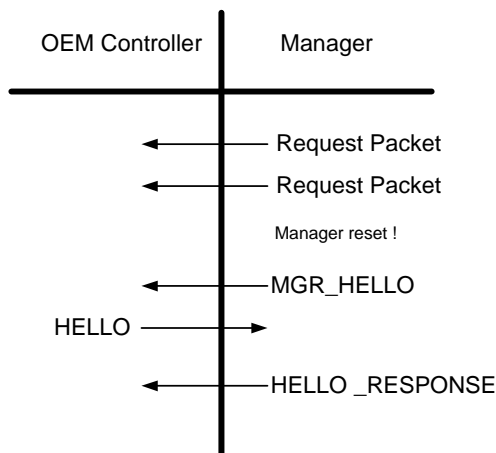


Figure 7 Hello Handshake Following a Manager Reset

OEM Controller State Machine

Figure 8 outlines the OEM controller state machine for session management. The following is a description of the three states represented in Figure 8.

- **Unconnected**—This is the initial state after reset or following a communication failure. To initiate a connection, the OEM controller sends a *HELLO* message and moves into the HELLO Sent state.
- **HELLO Sent**—The OEM controller stays in this state while waiting for a *HELLO_RESPONSE* message. Receiving a valid *HELLO_RESPONSE* puts the OEM controller into the Connected state. Timeout or errors will cause the OEM controller to go back into Unconnected state.
- **Connected**—The OEM controller stays in this state during the session. Errors or detecting a manager reset (receiving *MGR_HELLO*) causes the OEM controller to go into the Unconnected state.

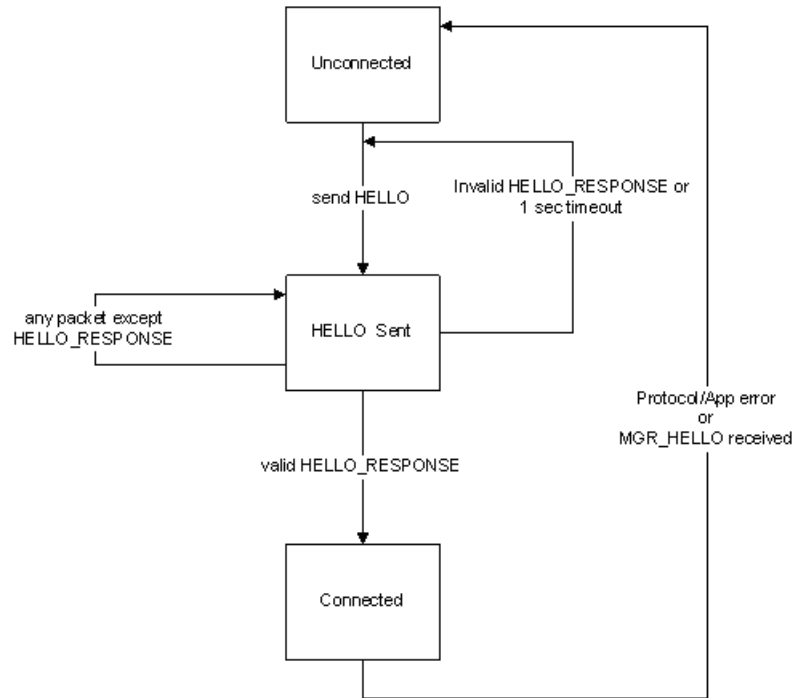


Figure 8 OEM Controller State Machine

Manager State Machine

Figure 9 outlines the manager session state machine. The following is a description of the three states represented in Figure 9.

- **Unconnected**—This is the initial state after a manager reset or following a communication failure. To notify the OEM controller of the manager reset, the manager sends a *MGR_HELLO* packet and moves into HELLO Sent state.
- **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 OEM controller causes the manager to send back a *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 OEM controller causes the manager to clear all of its settings and buffers, and re-enter the Connected state. Receiving a protocol error causes the manager to move into the Unconnected state.

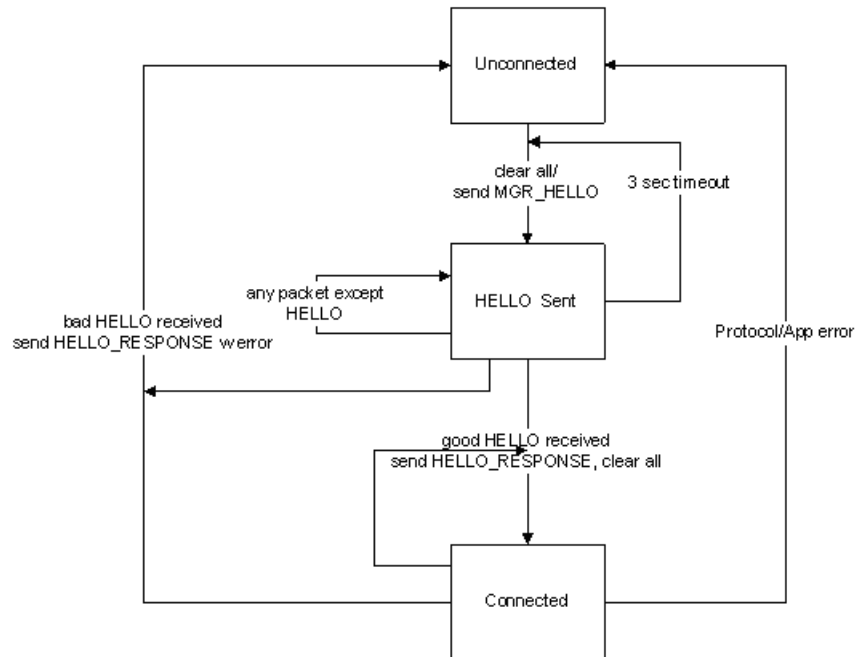


Figure 9 Manager State Machine

Session Establishment Commands

This section describes the establishment commands. Table 3 indicates the *type* value for these commands.

Table 3 Session Establishment Commands

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

MGR_HELLO

Table 4 MGR_HELLO

Field Name	Data Type	Value/Description
Control	Byte	0x00
Type	Byte	0x03
Sequence	Byte	Unused (set to 0x00)
Version	Byte	
Mode	Byte	0x00 = Operational mode 0x01 = Bootloader mode

version—The version of LM2650 Serial Protocol supported by this manager.

mode—Indicates whether the manager is in bootloader mode or operational mode

HELLO

HELLO packet is sent by the OEM controller to initiate new session with the manager.

Table 5 HELLO Packet

Field Name	Data Type	Value/Description
Control	Byte	0x00
Type	Byte	0x01
Sequence	Byte	Unused (set to 0x00)
Version	Byte	
Client sequence number	Byte	
Mode	Byte	0x00 = Operational mode 0x01 = Bootloader mode

version—The version of LM2650 Serial Protocol supported by the OEM controller. The manager checks this field to decide on compatibility with the OEM controller.

client sequence number—The unique number of this HELLO packet. The first packet from the OEM controller may start with any number, and each subsequent packet must carry must carry a successive sequence number. For more information, see “Sequence Number” on page 5.

mode—Indicates whether the manager is in bootloader mode or operational mode.

HELLO_RESPONSE

Table 6 HELLO_RESPONSE Packet

Field Name	Data Type	Value/Description
Control	Byte	0x00
Type	Byte	0x02
Sequence	Byte	Unused (set to 0x00)
Response code	Byte	Valid response codes: RC_OK RC_UNSUPPORTED VERSION RC_INVALID MODE See Table 7 for response code descriptions
Version	Byte	
Manager sequence number	Byte	
Client sequence number	Byte	
Mode	Byte	0x00 = Operational mode 0x01 = Bootloader mode

response code—The response code is used by the manager to indicate the result of session establishment. These codes are defined in Table 7.

version—The version of LM2650 Serial Protocol 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 (RC_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 (RC_UNSUPPORTED VERSION), and the version field will contain the supported version. Note that the version number reflects changes to packet structure, command fields, or command behavior. The manager currently supports only Serial Protocol version 2.

manager sequence number—This establishes the sequence number of the manager data stream. Subsequent messages sent by the manager will have incrementing sequence numbers. For more information, see “Sequence Number” on page 5.

client sequence number—This confirms the client sequence number received in the *HELLO* message.

mode—Indicates whether the manager is in bootloader mode or serial API mode.

Table 7 Response Codes for HELLO_RESPONSE

Hex Value	Name	Description
0x00	RC_OK	The session is successfully established.
0x01	RC_UNSUPPORTED VERSION	Unsupported protocol version.
0x02	RC_INVALID MODE	The <i>HELLO</i> was sent for wrong manager mode (for example, the manager is in bootloader mode and the <i>HELLO</i> was sent for the serial API mode).

Session Packet Processing

Once a session is established, each request packet must be explicitly acknowledged by the receiver with a response packet. Both sides can send data asynchronously from each other. That is, while waiting for a response packet, each device can receive a packet from the other side and should be ready to respond. Only one request at a time can be outstanding in each direction.

Note that Figure 10 shows requests from the manager to the OEM controller, however, requests are bidirectional.

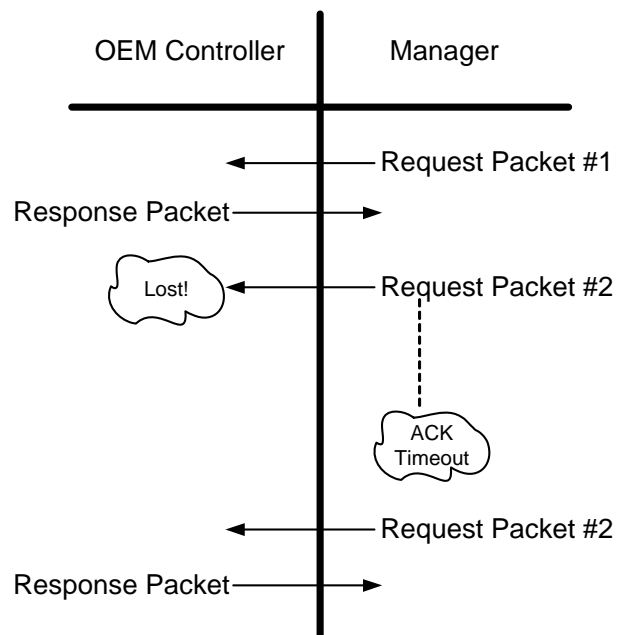


Figure 10 Communication Process

Session Packet Processing Guidelines

At the receiver, a request packet should be processed as follows:


- ① Compare the sequence number of the request packet with the previous sequence number from the sender.
- ② If the two sequence numbers match, the request packet is considered a duplicate. The receiver sends a copy of previous response packet and drops the request packet.

If the sequence number of the request packet equals the previous sequence number plus one, the request packet should be processed and a response sent back. *A copy of last response packet is always stored until the next request packet is received.*

If request packet has any other sequence number, the receiver should explicitly drop the packet.

At the sender, the following process should be used to send a request packet:

- ❶ Assign the next sequence number to the packet.
- ❷ Send the packet.
- ❸ Set the timeout for receiving the response packet (one second is recommended).
- ❹ If a response packet with the correct sequence number is received, the packet transmission is complete. (The response to a request must contain the sequence number of the request.)
- ❺ If the response packet timeout has been exceeded, retry steps 2-5 (retrying three times is recommended).
- ❻ If a maximum number of retries has been reached, the session should be considered as “failed” and a new handshake must be established.

 **Note:** The sequence number counters maintained by the OEM controller and the manager are independent of one another. However, the sequence number of a response packet must match that of its corresponding request packet.

Example of Failed Communication

The following example illustrates what happens when a request or response packet is lost during a data exchange.

- First, the *getTime* command with sequence number 5 is sent, and the manager sends the *getTime* response with sequence number 5.
- Next, the OEM controller sends a *sendRequest* request command with sequence number 6, and the packet is lost. Following a timeout, the OEM controller resends the *sendRequest* request packet with sequence number 6. Now, the packet is received by the manager, but the response packet with sequence number 6 gets lost.
- Finally, the OEM controller retries the *sendRequest* request command 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 OEM controller receives it.

End-to-end Wireless Communication

Users may choose reliable or best-effort communication for serial transmissions between the OEM controller and the sensor microprocessor. The LM2610 manager supports reliable communication in the *downstream* direction. Downstream communication refers to packets sent from the OEM controller to the sensor microprocessor.

Reliable Wireless Communication

In end-to-end reliable communication, packets sent wirelessly are acknowledged end-to-end, providing confirmation to the application layer on the OEM controller 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 OEM controller uses the *sendRequest* command with the *isReliable* field set. When the manager receives the response from the network, it forwards it to the OEM controller 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 is not set, no *serial data notification* is sent to the OEM controller. 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.

Access Control List

The access control list (ACL) specifies the list of motes that will be allowed to join the LM2650 network. The ACL is managed by the OEM controller, which uses the *setAclEntry* command to add motes and the *delAclEntry* command to remove motes from the ACL. If the ACL contains no entries, the LM2650 manager will allow any mote to join that has the correct common join key and network ID (assuming the mote and manager are running compatible software). Although the default common join key is sufficient for demonstration networks, the ACL should be used for product release. The ACL becomes active when there is at least one mote entry in the list.

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 “Events” on page 42).

When adding a mote to an ACL network, you must first add the mote’s MAC address 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.

Network Bandwidth Control

The LM2650 manager is able to accommodate the network performance requirements of a wide range of applications by providing two *bandwidth profiles*. The bandwidth profile defines the upstream/downstream frame size (number of timeslots) the network will use for network formation and network operation. Each profile is associated with a different set of performance parameters representing a trade-off between network speed and power consumption (see Table 8). The bandwidth profile is set using the *setNetwork* command (see “setNetwork” on page 37).

Table 8 Bandwidth Profiles

Profile Name	Description
P1 (default)	For fast network build and medium speed network operation. <ul style="list-style-type: none"> • Typical 90-second reporting per mote • Lower latency than P2
P2	For fast network build and slow network operation. <ul style="list-style-type: none"> • Lower power consumption than P1 • Slower reporting per mote than P1

Response Codes

The following table describes the response codes that may be returned in the command response packet.

Table 9 Response Codes

Hex Value	Name	Description
0x00	RC_OK	The application has processed the command successfully.
0x01	RC_INVALID_CMD	The command is invalid.
0x02	RC_INVALID_ARG	The command argument is invalid.
0x03	RC_CMD_FAILED	The bootloader command failed.
0x04	RC_INVALID_STATE	The bootloader is not ready to process the command or the command failed.
0x0B	RC_END_OF_LIST	End of list reached.
0x0C	RC_NO_ROOM	The maximum number of items has been reached.
0x0D	RC_IN_PROGRESS	Operation is in progress
0x0E	RC_NAK	The manager has received the packet but is not able to process it at this time. The OEM controller should retry sending the packet.
0x0F	RC_WRITE_ERROR	The flash write failed.

Commands

This section provides detailed information for each serial API command. *All commands are immediately effective unless otherwise noted.* 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 *delAclEntry* command (Figure 11) contains a total of 11 bytes. Note that the packet header fields appear shaded and the packet payload is unshaded. Refer to the section “Byte Ordering and Data Types” on page 4 for details regarding the structure of each data type.

Table 10 delAclEntry Request

	Field Name	Data Type	Value/Description
1 byte	Control	Byte	0x02
1 byte	Type	Byte	0x29
1 byte	Sequence	Byte	
8 bytes	MAC address	Byte [8]	

Figure 11 Example Packet

Table 11 provides a short description of each command and the command type.

Table 11 Command Summary

Type	Command	Description
Sending and Receiving Data		
0x16	subscribe	Subscribes to notifications to receive data, network events, and alarms.
0x20	sendRequest	Sends serial data to a specific mote.
Manager/Network Settings and Descriptions		
0x17	getTime	Requests that the manager send the current time and corresponding ASN (absolute slot number). The time is sent in a notification message.
0x18	getSystem	Returns various system information for the manager.
0x19	getNetwork	Returns various network information.
0x1A	setNetwork	Sets network configuration parameters.
0x1B	getMote	Returns information for a given mote.
Security		
0x28	getNextAclEntry	Returns information about the next mote on the access control list (ACL).
0x27	setAclEntry	Adds a mote to the access control list (ACL).

Table 11 Command Summary (Continued)

Type	Command	Description
0x29	delAclEntry	Removes a mote from the access control list (ACL).
0x22	exchangeNetworkId	Causes the manager to distribute a new network ID to all motes in the network.
Network Performance		
0x1C	getNetStatistics	Returns data reliability, data latency, and path stability statistics for a given period or for the lifetime of the network (lifetime average).
0x1D	getMoteStatistics	Returns mote statistics for a given period or for the lifetime of the mote. Mote statistics include reliability, average latency, number of joins, and charge consumption.
0x1E	getNextPathStatistics	Returns path statistics for a given period or for the lifetime of the path.
0x1F	clearStatistics	Clears the current set of network, mote, and path statistics.
Administrative and Maintenance Commands		
0x24	radioTestTx	Places the manager in radio test mode and executes the radio transmission test.
0x25	radioTestRx	Places the manager in radio test mode and executes the radio reception test.
0x26	radioTestStats	Retrieves statistics from the most recent radio test.
0x15	reset	The reset command resets the system or a mote.
0x2A	pingMote	Sends a packet to a mote requesting a response.
0x2B	getLog	Retrieves logs from the manager or an individual mote

clearStatistics

Description

The *clearStatistics* command clears the current set of network, mote, and path statistics. This command does not clear lifetime statistics.

Request Packet

Table 12 clearStatistics Request


Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x1F
Sequence	Byte	

Response Packet**Table 13 clearStatistics Response**

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x1F
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK See Table 9 for response code descriptions.

delAclEntry**Description**

The *delAclEntry* command removes a mote (or all motes) from the access control list (ACL). If a mote is connected to the network when it is deleted from the ACL list, it will not be able to rejoin the network the next time it resets.

 **Note:** If the ACL contains no entries, the manager will use the default common join key for network security. Although this may be appropriate for demonstration networks, installed network applications should use the ACL. For more information, see “Access Control List” on page 15.

Request Packet**Table 14 delAclEntry Request**

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x29
Sequence	Byte	
MAC address	Byte [8]	MAC address of mote to remove

MAC address—The MAC address of the mote to be removed from the ACL. To remove all motes from the ACL (clear the ACL), use a MAC address of FF-FF-FF-FF-FF-FF-FF-FF.

Response Packet**Table 15 delAclEntry Response**

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x29
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_END_OF_LIST RC_WRITE_ERROR See Table 9 for response code descriptions.

exchangeNetworkId

Description

The *exchangeNetworkId* command triggers the manager to distribute a new network ID to the motes. Note that the *callback ID* contained in the *exchangeNetworkId* response packet is included in the *command finished* notification that is sent to the OEM controller when the network ID exchange is completed. The new network ID is used after the system is restarted using the *reset* command.

Request Packet

Table 16 exchangeNetworkId Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x22
Sequence	Byte	
New network ID	ShortInt	

Response Packet

Table 17 exchangeNetworkId Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x22
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_IN_PROGRESS RC_NAK RC_WRITE_ERROR See Table 9 for response code descriptions.
Callback ID	LongInt	

getLog

Description

The *getLog* command allows the retrieval of logs from the LM2650 manager or an individual mote. The log contents are sent from the manager to the OEM controller in a log notification (see “Log Notification Packet” on page 41).

Request Packet

Table 18 getLog Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x2B
Sequence	Byte	
MAC address	Byte[8]	

MAC address—To retrieve the manager log, set the MAC address byte to 00-00-00-00-00-00-00-00. To retrieve the mote log, set this byte to the mote MAC address.

Response Packet

Table 19 getLog Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x2B
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_END_OF_LIST RC_NO_ROOM RC_IN_PROGRESS See Table 9 for response code descriptions.

getMote

Description

The *getMote* command returns information about a given mote.

Request Packet

Table 20 getMote Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x1B
Sequence	Byte	
MAC address	Byte[8]	
Next flag	Byte	0x00 = Find mote with this MAC address 0x01 = Find next mote

MAC address— To retrieve the first mote in the list, set the MAC address byte to 00-00-00-00-00-00-00-00 and the Next flag to 0x01.

Next flag—Indicates whether the MAC address refers to the requested mote or to the next mote in manager's memory.

Response Packet**Table 21 getMote Response**

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x1B
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_END_OF_LIST See Table 9 for response code descriptions.
MAC address	Byte[8]	
Hardware model	Char[17]	
Hardware revision	Char[17]	
Software revision	Char[17]	
isAccessPoint	Byte	0x00 = Mote is not an access point 0x01 = Mote is an access point
State	Byte	See Table 22
Join time (ASN)	Byte[5]	
Voltage	FixedPoint	
Temperature	FixedPoint	
Number of neighbors	ShortInt	

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

hardware model—The hardware model number.

hardware revision—The hardware revision number.

software revision—While the mote is starting up, the software 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).

isAccessPoint—Indicates that this mote is an access point. The access point is physically located on the LM2650 manager board.

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 all zeros.

voltage—Contains the reading from the last battery measurement.

temperature—The temperature of the mote, in Celsius.

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

Mote States

Table 22 Mote States

Value	Mote State	Description
0x00	LOST	Indicates the mote was previously in the network but has dropped out of the network.
0x01	NEGOTIATING1	The mote is in the process of joining the network and is exchanging the security handshake with the manager. The mote may be joining the network for the first time or re-joining the network from the LOST state.
0x02	NEGOTIATING2	The mote has been authenticated (security handshake was completed) and is in the process of receiving links to one parent.
0x03	CONNECTED	The mote has received links to one parent and is in the process of receiving the additional links it needs to become fully operational.
0x04	OPERATIONAL	The mote has been assigned all of its links and is operational and ready to be used.

getMoteStatistics

Description

The *getMoteStatistics* command returns the current set of statistics or lifetime statistics for a specified mote. Current statistics is the set gathered since the *clearStatistics* command was last executed. Lifetime statistics is aggregated over the lifetime of the mote.

Request Packet

Table 23 getMoteStatistics Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x1D
Sequence	Byte	
MAC address	Byte [8]	
Statistics set	Byte	0x00 = Current statistics 0x01 = Lifetime statistics

statistics set—Specifies the statistics set (current or lifetime statistics).

Response Packet**Table 24 getMoteStatistics Response**

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x29
Sequence	Byte	0x1D
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG RC_END_OF_LIST See Table 9 for response code descriptions.
Requested statistics set	Byte	0x00 = Current statistics 0x01 = Lifetime statistics
Start time of interval (as ASN)	Byte[5]	
Reliability	FixedPoint	
Latency (msec)	LongInt	
NumJoins	ShortInt	
Charge consumption	LongInt	

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

requested statistics set—The set of statistics (current or lifetime) specified in the request packet.

start time of interval—The start time of the statistics set.

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.

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

charge consumption—The cumulative millicoulombs consumed by mote.

getNetStatistics

Description

The *getNetStatistics* command returns the current set of statistics or lifetime statistics for the network. Current statistics is the set gathered since the *clearStatistics* command was last executed. Lifetime statistics is the average over the lifetime of the network.

Request Packet

Table 25 getNetStatistics Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x1C
Sequence	Byte	
Statistics set	Byte	0x00 = Current statistics 0x01 = Lifetime statistics

Response Packet

Table 26 getNetStatistics Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x1C
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG See Table 9 for response code descriptions.
Requested statistics set	Byte	0x00 = Current statistics 0x01 = Lifetime statistic
Start time of interval (as ASN)	Byte[5]	
Reliability	FixedPoint	
Stability	FixedPoint	
Latency (msec)	LongInt	

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

requested statistics set—The set of statistics (current or lifetime) specified in the request packet.

start time of interval—The start time of the requested statistics set.

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.

getNetwork

Description

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

Request Packet

Table 27 getNetwork Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x19
Sequence	Byte	

Response Packet

Table 28 getNetwork Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x19
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK See Table 9 for response code descriptions.
Network ID	ShortInt	
Access point power amplifier	Byte	0x00 = Off 0x01 = On
Number of motes	ShortInt	
Bandwidth profile	Byte	See Table 8.

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

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

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

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

bandwidth profile—The bandwidth profile determines the network performance parameters. See “Network Bandwidth Control” on page 16.

getNextAclEntry

Description

The *getNextAclEntry* command returns the MAC address and join key for the next mote entry on the access control list (ACL) following the mote you specify. To retrieve the first entry in the ACL list, set the *MAC address* field to FF-FF-FF-FF-FF-FF-FF-FF.

Request Packet

Table 29 getNextAclEntry Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x28
Sequence	Byte	
MAC address	Byte [8]	

Response Packet

Table 30 getNextAclEntry Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x28
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_END_OF_LIST See Table 9 for response code descriptions.
MAC address	Byte[8]	
Join key	Char[16]	

getNextPathStatistics

Description

The *getNextPathStatistics* command returns the current set of statistics or lifetime statistics for the next path entry in the list following the path you specify.

Request Packet

Table 31 getNextPathStatistics Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x1E
Sequence	Byte	
Path ID	LongInt	
Statistics set	Byte	0x00 = Current statistics 0x01 = Lifetime statistics

path ID—The ID of the path for which statistics is being requested. To retrieve statistics for the first path in the list, use a path ID of zero.

statistics set—Indicates whether current or lifetime statistics is requested. Current statistics is the set gathered since the *clearStatistics* command was last executed. Lifetime statistics is the average over the lifetime of the path.

Response Packet

Table 32 getNextPathStatistics Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x1E
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG RC_END_OF_LIST See Table 9 for response code descriptions.
Path ID	LongInt	
Requested statistics set	Byte	0x00 = Current statistics 0x01 = Lifetime statistic
Start time of interval	Byte[5]	
MAC address mote A	Byte[8]	
MAC address mote B	Byte[8]	
Average RSSI mote A to B	Byte	RSSI is a signal, in dBm. Reported as a signed byte.
Average RSSI mote B has to neighbor A	Byte	RSSI is a signal, in dBm. Reported as a signed byte.
Total number of links	ShortInt	
Stability	FixedPoint	

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

path ID—The ID of the path for which statistics is being provided.

requested statistics set—Indicates whether current or lifetime statistics is provided.

start time of interval—The start time, as ASN.

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

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

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

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

total number of links—The total number of links between mote A and mote B.

stability—The ratio of successful packet transmissions to the total number of packet transmissions on this path (0-100).

getSystem

Description

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

Request Packet

Table 33 getSystem Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x18
Sequence	Byte	

Response Packet

Table 34 getSystem Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x18
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK See Table 9 for response code descriptions.
Software version	Char[17]	
Serial number	Char[17]	

getTime

Description

The *getTime* command triggers a *network time* notification containing the network uptime and the ASN. Note that the *callback ID* contained in the *getTime* response packet is included in the *network time* notification that is sent to the OEM controller. The network uptime is the number of seconds the manager software has been running and the ASN (absolute slot number) is the number of 10 ms timeslots since the system was reset.

Request Packet

Table 35 getTime Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x17
Sequence	Byte	

Response Packet**Table 36** getTime Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x17
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_IN_PROGRESS See Table 9 for response code descriptions.
Callback ID	LongInt	

pingMote

Description

The *pingMote* command sends a packet to a mote requesting a response. When the manager receives an acknowledgement from the mote (or if the ping request times out), it sends a *ping reply* notification to the OEM controller. If the ping request times out, the notification contains a round-trip latency of -1. Note that the *callback ID* contained in the *pingMote* response packet is included in the *ping reply* notification.

Request Packet**Table 37** pingMote Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x2A
Sequence	Byte	
MAC address	Byte[8]	

Response Packet**Table 38** pingMote Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x2A
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_END_OF_LIST RC_NO_ROOM RC_IN_PROGRESS See Table 9 for response code descriptions.
Callback ID	LongInt	Callback ID is only generated when the response code is 0x00.

radioTestRx

Description

The *radioTestRx* command suspends normal network operation, places the manager in radio test mode, clears all previously collected statistics, and executes a radio reception test for the specified channel and duration. Use this command to test the manager's radio for FCC certification purposes.

During the test, the access point mote keeps statistics of the number of packets received (with and without error). The test results may be retrieved using the *radioTestStats* command. After radio testing and statistics retrieval is completed, issue the *reset* command and reset the system to return to normal network operation.

Request Packet

Table 39 radioTestRx Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x25
Sequence	Byte	
Channel	Byte	RF channel 0-15
Duration	ShortInt	Test duration in seconds

channel—The channel to be used for the test.

duration—Duration of the test, in seconds.

Response Packet

Table 40 radioTestRx Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x25
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG RC_IN_PROGRESS See Table 9 for response code descriptions.

radioTestStats

Description

The *radioTestStats* command retrieves statistics from the most recent radio test.

Request Packet

Table 41 radioTestStats Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x26
Sequence	Byte	

Response Packet

Table 42 radioTestStats Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x26
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_CMD RC_IN_PROGRESS See Table 9 for response code descriptions
Number of packets successfully received	ShortInt	
Number of packets not received	ShortInt	

radioTestTx

Description

The *radioTestTx* command suspends normal network operation, places the manager in radio test mode, and then executes a radio transmission test. Use this command to test the manager's radio for FCC certification purposes.

Each packet transmission is 128 bytes in length. As shown in Figure 12, byte 0 contains the packet length (excluding the length parameter itself), which is 127 bytes. Bytes 1 and 2 contain the *sequence number* that increment with every packet transmitted. Bytes 3-125 contain a big-endian counter (from 0-122) that increments with every byte. The interpacket delay is 100 ms. Note that if *number of packets* is set to 0, the access point mote generates an unmodulated test tone on the selected channel.

To return to normal network operation, issue the *reset* command and reset the system. If you were testing using an unmodulated test tone, the *reset* command will stop the access point mote from generating the unmodulated test tone.

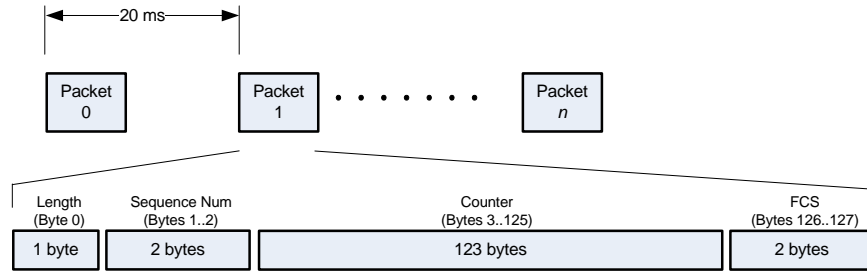


Figure 12 radioTestTx Packet Format

Request Packet

Table 43 radioTestTx Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x24
Sequence	Byte	
Channel	Byte	RF channel 0-15
Transmit power	Byte	-2 = Power amplifier off 8 = Power amplifier on
Number of packets	ShortInt	

channel—The channel to be used for the transmission test.

transmit power—Sets the transmit power amplifier on or off for the transmission test.

number of packets—The number of packets to transmit.

Response Packet

Table 44 radioTestTx Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x24
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG RC_IN_PROGRESS See Table 9 for response code descriptions

reset

Description

The *reset* command resets the system or a mote. A system reset restarts the manager software processes, resets all statistics for the network, and resets the manager's wireless connection and causes the network to reform. A mote reset causes a specific mote to reboot and rejoin the network. Keep in mind that the reset may not succeed, given that the command is sent as a best-effort packet.

Request Packet

Table 45 reset Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x15
Sequence	Byte	
Reset type	Byte	0x00 = System 0x02 = Mote
MAC address	Byte[8]	

reset type—Indicates whether the system, network, or mote is to be reset.

MAC address—Used only when reset type is 0x02 (mote). Specifies the mote to be reset (do not specify the access point mote).

Response Packet

Table 46 reset Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x15
Sequence	Byte	
Response code	Byte	RC_OK RC_END_OF_LIST (mote with specified MAC address was not found or was not in operational state) RC_NAK RC_INVALID_ARG See Table 9 for response code descriptions.
MAC address	Byte[8]	

MAC address—If a mote reset was specified in the request command, the response returns the MAC address of the mote. If the system is reset, the *MAC address* field is empty.

sendRequest

Description

The *sendRequest* command sends a request packet to a specific mote. Note that the *sendRequest* request is the only command that contains an extended header that adds four control bytes inside the packet.

Request Packet

Table 47 sendRequest Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x20
Sequence	Byte	
MAC address	Byte [8]	
Packet priority	Byte	0x00 = Low 0x01 = High
isReliable	Byte	0x00 = Best effort 0x01 = Reliable
Serial data length	Byte	Length of the serial data field (up to 94 bytes)
Reserved	Byte	0x00
Reserved	Byte	0x00
Reserved	Byte[2]	0xFC12
Serial data	Byte []	Serial data payload

MAC address—If this field is set to FF 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 0x00 (best-effort).

packet priority—Sets the priority of this packet.

isReliable—Allows the OEM controller to specify how the packet will be sent through the network. There are two options:

- Setting *isReliable* to 0x01 (reliable) ensures end-to-end acknowledgment of packet arrival at the mote. The *callback ID* provided in the response packet (see Table 48) will be included in the corresponding *serial data* notification (see Table 57).
- Setting *isReliable* to 0x00 (best-effort) still provides link-level acknowledgments, but no end-to-end acknowledgement. In this case, the *callback ID* provided in the response packet is zero, and no *serial data* notification will be sent.

serial data length—This is the length of the *serial data* field (up to 94 bytes).

serial data—This field contains the serial data payload (up to 90 bytes). For information on payload compatibility with future Dust Networks products contact Dust Networks Support.

Response Packet**Table 48** sendRequest Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x20
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG RC_END_OF_LIST RC_NAK See Table 9 for response code descriptions.
MAC address	Byte[8]	
Callback ID	LongInt	

setAclEntry**Description**

The *setAclEntry* command adds a new mote entry to the access control list (ACL) or updates an existing ACL entry. For more information about the ACL, see “Access Control List” on page 15.

Request Packet**Table 49** setAclEntry Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x27
Sequence	Byte	
MAC address	Byte[8]	
Join key	Byte[16]	

Response Packet**Table 50** setAclEntry Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x27
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_NO_ROOM RC_WRITE_ERROR See Table 9 for response code descriptions.

setNetwork

Description

The *setNetwork* command allows you to change network parameters. The manager must be reset for the new parameters to take effect. The response shows the new parameter settings.

For field descriptions, see “getNetwork” on page 26.

Request Packet

Table 51 setNetwork Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x1A
Sequence	Byte	
Network ID	ShortInt	
Access point power amplifier	Byte	0x00 = Off 0x01 = On
Bandwidth Profile	Byte	1 = P1 2 = P2 See Table 8 for bandwidth profile descriptions.

Response Packet

Table 52 setNetwork Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x1A
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_WRITE_ERROR See Table 9 for response code descriptions.
Network ID	ShortInt	
Access point power amplifier	Byte	0x00 = Off 0x01 = On
Number of motes	ShortInt	
Bandwidth Profile	Byte	1 = P1 2 = P2 See Table 8 for bandwidth profile descriptions.

subscribe

Description

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

Request Packet

Table 53 Subscribe Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x16
Sequence	Byte	
Filter	Byte[4]	See Table 54

Each subscription request overwrites the previous one. For example, if the OEM controller is subscribed to serial data and then wants system events as well, the OEM controller should send a *subscribe* command with both the *serial data* and *system events* flags set in the *filter* field (see Tables 54). To clear all subscriptions, the OEM controller should send a *subscribe* command with the *filter* field set to zero.

Table 54 Filter Masks

Notification	Bitmask
Serial data	0x0000 0001
System events	0x0000 0x02
Log messages	0x0000 0x04

Response Packet

Table 55 Subscribe Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x16
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG See Table 9 for response code descriptions.

Remarks

After the *subscribe* response, asynchronous packets will be sent by the manager to the serial interface. These packets will contain the data notifications or events requested by the OEM controller. All notifications require an acknowledgement, and subsequent notification packets will be queued while waiting for the acknowledgement.

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

Notifications

Notifications are asynchronous messages from the LM2650 manager to the OEM controller, 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 manager sends a notification message to its OEM controllers with the contents of the serial packet. Event notifications capture occurrences such as the network being reset, or a new mote joining the network. Table 56 summarizes the types of notifications. The following sections describe each notification in detail.

Table 56 Notifications

Notification	Description	Notification Type
Serial Data Notification	Contains serial data sent from a mote in the wireless network	0x00
Event Notifications	See “Events” on page 42.	0x01
Log Notifications	Information used for network troubleshooting.	0x02

The OEM controller uses the *subscribe* command to select the types of notifications it wants to receive. The OEM controller 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 OEM controller that match the requested event types. Since at most one packet can be outstanding in the link layer, the notification packets are interleaved with acknowledgements and command responses.

All notification packets have the packet type 0x14 (notification type). The notification structure is described below for each type of notification.

If the OEM controller receives a notification type that it does not understand, it must send a response to the notification even though it does not use the notification.

Notification Packet Format

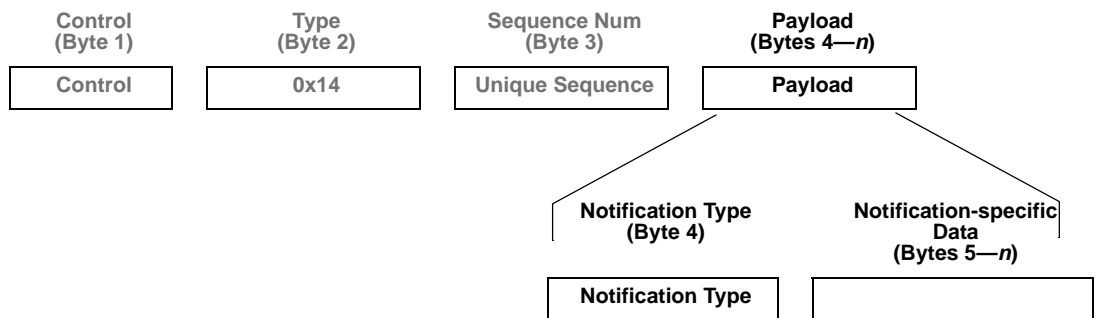


Figure 13 Notification Packet Format

Notification Details

The following sections provide a byte-level description of each notification packet.

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.

Request Packet

Table 57 Serial Data Notification Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x14
Sequence	Byte	
Notification type	Byte	0x00
Packet timestamp	Byte[5]	
MAC address	Bytes[8]	
Callback ID	LongInt	
Reserved	Byte	0x00
Reserved	Byte	0x00
Reserved	Byte[2]	0xFC12
Serial data	Byte []	Serial data payload

packet timestamp—The time that the packet was generated (as ASN).

MAC address—The MAC address of the mote that generated the notification.

callback ID—This field contains the callback ID provided in the response to the *sendRequest* request that triggered the *serial data notification* if the *sendRequest* request was sent as a reliable packet.

serial data—The serial data payload (up to 90 bytes). The serial data is a variable length array of binary data. For information on payload compatibility with future Dust Networks products contact Dust Networks Support.

Response Packet

Table 58 Serial Data Notification Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x14
Sequence	Byte	

Event Notification Packet

Events describe system-level changes and changes in the network topology. For a description of the event types, see “Events” on page 42.

Request Packet

Table 59 Event Notification Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x14
Sequence	Byte	
Notification type	Byte	0x01
Event payload		See “Events” on page 42.

Response Packet

Table 60 Event Notification Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x14
Sequence	Byte	

Log Notification Packet

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

Request Packet

Table 61 Log Notification Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x14
Sequence	Byte	
Notification type	Byte	0x02
Packet timestamp (as ASN)	Byte[5]	
Network uptime	LongInt	
Log message	Char []	Maximum 115 bytes.

Response Packet

Table 62 Log Notification Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x14
Sequence	Byte	

Events

This section describes the packet payload for each type of event notification. The packet structure for event notifications is described in “Event Notification Packet” on page 41.

Table 63 Events

Event	Description	Event Type
Mote Reset	A mote was reset.	0x00
Command Finished	An <code>exchangeNetworkId</code> , or <code>exchangeSessionKey</code> command has completed execution.	0x02
Mote Join	A mote joined the network.	0x03
Mote Operational	A new mote was configured.	0x04
Mote Lost	A mote is no longer communicating in the network.	0x05
Network Time	Contains the network uptime (in response to a <code>getTime</code> command).	0x06
Ping Reply	A reply was received from a mote ping.	0x07
Brownout	A drop in the battery supply voltage was detected by the LM2650 manager.	0x08

Mote Reset

This notification is sent when a user-initiated reset is executed by the manager.

Table 64 Mote Reset Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63
MAC address	Byte[8]	

Command Finished

This notification is sent when a command execution (such as `exchangeNetworkId`) is completed. The `callback ID` is the callback ID that was returned in the response packet of the corresponding command.

Table 65 Command Finished Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63
Callback ID	LongInt	

Mote Join

This notification is sent when a mote joins the network and the mote state transitions to the mote state “Negotiating 1” (see Table 22).

Table 66 Mote Join Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63
MAC address	Byte[8]	

Mote Operational

This notification is sent when a mote that joins the network becomes operational (see Table 22) and is ready to be used.

Table 67 Mote Operational Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63
MAC address	Byte[8]	

Mote Lost

This notification is sent once when the mote changes to the Lost state, which may occur when a mote becomes unreachable through the network.

Table 68 Mote Lost Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63
MAC address	Byte[8]	

Network Time

This notification is sent in response to a *getTime* request. It contains the network uptime (the number seconds the manager software has been running) and the ASN (absolute slot number), which is the number of 10 ms timeslots since the network started up. The *callback ID* is the callback ID that was returned in the response packet associated with the *getTime* request.

Table 69 Network Time Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63
Callback ID	LongInt	
Network uptime (seconds)	LongInt	
ASN	Byte[5]	

Ping Reply

This notification is sent when a reply is received from a mote ping. The *callback ID* is the callback ID that was returned in the response packet associated with the *pingMote* request. A latency value of -1 indicates that the ping request timed out and no response was received from the mote.

Table 70 Ping Reply Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63
Callback ID	LongInt	
MAC address of mote pinged	Byte[8]	
Latency (msec)	LongInt	

Brownout

This notification is sent when the LM2650 manager detects a drop in its battery supply voltage. When a “brownout” condition exists, you cannot perform software updates or set network configuration parameters (*setNetwork* command) or change the access control list (*setAclEntry* command).

Table 71 Brownout Event Payload

Field Name	Data Type	Value/Description
Event ID	LongInt	
Event type	Byte	See Table 63

Forward Compatibility

The following guidance should be followed to ensure compatibility with software revisions and future Dust Networks products:

- Receive buffers should be sized to receive the maximum packet size documented in the API (128 bytes).
- If a notification with extra fields is received, the extra fields should be ignored.
- If a longer than expected response is received, the extra fields should be ignored.
- If an unknown notification is received, it should be acknowledged and ignored.

Bootloader API



This appendix describes how to use LM2650 bootloader API commands to access the manager's flash and update software on the manager or access point.

Overview

The LM2650 bootloader uses a serial protocol that is a packet-based communication protocol over an asynchronous serial port. The protocol runs at 115 kbps, 8 bits, no parity, 1 stop bit. RTS/CTS hardware handshaking is used, as described in the product datasheet.

When the manager is powered on or reset, the bootloader launches and sends four *MGR_HELLO* packets to the OEM controller (once every 3 seconds). If the OEM controller responds to any of these *MGR_HELLO* packets with *HELLO*, a bootloader session is established. However, if the OEM controller does not respond, the bootloader loads the manager software.

When a bootloader session is established, the OEM controller can begin the file upload process by sending a *prepare* command which provides information required for the file transfer. After receiving a response to the *prepare* command, the OEM controller issues multiple *write* commands to send blocks of data into the manager flash or the AP flash, (each *write* command sends one block of data). After the upload is complete, the OEM controller uses the *verify* command to verify the upload and then uses the *boot* command to start the manager. Note that if a bootloader process fails, the OEM controller simply resends the *prepare* command and restarts the upload from the beginning.

The file upload process is illustrated in Figure 14. The following sections provide detailed information about the update file format and the bootloader API commands.

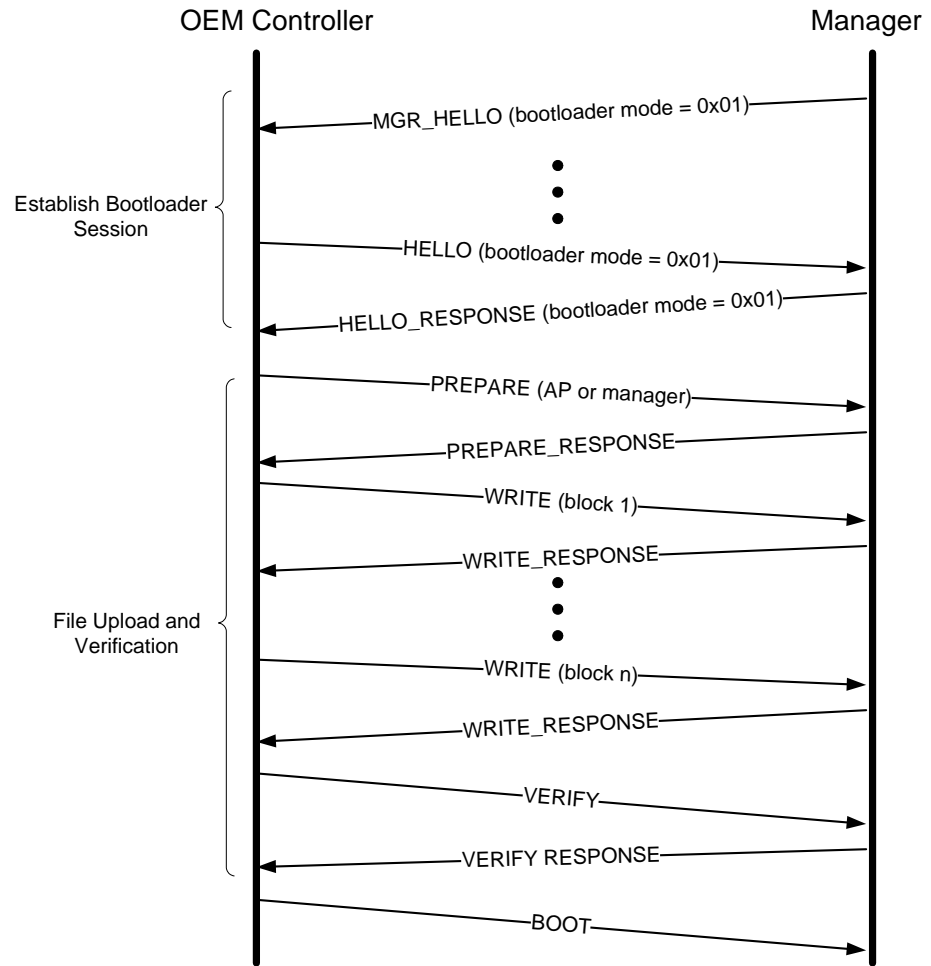


Figure 14 File Upload Process

Software Update File Format

Dust Networks provides two types of software update files—one for updating the manager application code, and one for updating the access point (AP). The software update files contain the application image along with specific header information that needs to be copied into the *prepare* command to initiate the software update. Table 72 shows the structure of the software update file, with the shaded fields indicating the header information. The *length* field (byte 1) specifies the number of following bytes that need to be copied into the *prepare* command. The *control block* field include the *block size*, which specifies the size of the block of data that should be used by the *write* command.

Table 72 Software Update File Format

Field Name	Data Type	Value/Description
Length	Byte	
Control block	Byte[2]	Reserved
	Byte	Block size
	Byte[]	Image-specific parameters for the prepare command
Data	Byte[]	The application image.

Establishing a Bootloader Session

Before using the bootloader API, the OEM controller needs to establish a bootloader session with the manager by exchanging *HELLO* packets. The session establishment commands (*HELLO_MGR*, *HELLO*, and *HELLO_RESPONSE*) are described in detail in “Session Establishment Commands” on page 10.

To establish a bootloader session with the manager:

- ❶ Generate periodic *HELLO* packets (once per second is the recommended interval) with the *mode* set for bootloader. For each new packet generated, increment the *client sequence number* (start with *client sequence number* 0).
- ❷ If a valid *HELLO_RESPONSE* packet is received (*response* code is *RC_OK* and the *client sequence number* and *mode* matches that in the *HELLO* packet just sent), the session is established and data can start flowing.
- ❸ If an invalid *HELLO_RESPONSE* packet is received (response code is not *RC_OK* or the *mode* does not match the *HELLO* packet), the OEM controller can choose to adjust the *HELLO* packet parameters (for example, the *version* field), and resend the *HELLO* packet.
- ❹ If no *HELLO_RESPONSE* packet is received, the OEM controller should continue resending *HELLO* until a response is received.
- ❺ Any other packet (including *MGR_HELLO*) received in response to *HELLO* packet must be dropped.

Upon completing *HELLO* packet exchange both the OEM controller and the manager should do the following.

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

How to Use Bootloader Commands

Once a session is established with the bootloader API (see the previous section), follow these steps to update the manager or AP software image.

To update the manager or AP software image:

- ❶ Create the *prepare* command.
- ❷ Send the *prepare* command to the manager and wait for the command response (0.5 second timeout).
- ❸ Send the *write* command with a block of data from image and wait for the command response (0.5 second timeout).
- ❹ If there is more data to send, repeat step 3 and write the next block of data.
- ❺ Send the *verify* command and wait for the command response (4 second timeout).

Bootloader Commands

Table 73 provides a short description of each command and the command *type*.

Table 73 Bootloader Command List

Type	Command	Description
0x04	prepare	Initializes a file upload to the manager or the access point.
0x05	write	Writes one block of the upload file into the manager or AP flash.
0x06	read	Downloads data from the manager's flash.
0x07	verify	Verifies the software image.
0x08	boot	Verifies the manager software and starts the manager application.
0x09	reset	Stops the current bootloader process and resets the manager.
0x0A	shutDown	Places the manager in low power sleep mode. The network is disconnected when manager is shut down.

prepare

Description

The *prepare* command initializes a file upload. To construct the *prepare* command, the application should read the first byte (*length* field) from the software update file and copy the indicated number of following bytes into the *control block* field in the prepare request packet. See “Software Update File Format” on page 47 for details about the software update file.

Request Packet

Table 74 prepare Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x04
Sequence	Byte	
Control block	Byte[]	

Response Packet

Table 75 prepare Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x04
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG RC_INVALID_STATE (the AP is not ready) See Table 9 for response code descriptions.

write

Description

The *write* command uploads one block of a file into the manager or AP flash. The *block number* indicates which segment of the file is being uploaded. The first block number is zero, and should be incremented with each write request. The data is the raw data being uploaded. The length of the data must match the block size indicated in the *prepare* command (in the *control block* field), and corresponds to byte 4 of the software update file (see Table 72). Note that the last block sent can be smaller.

Request Packet**Table 76 write Request**

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x05
Sequence	Byte	
Block number	ShortInt	
Data	Byte[]	

Response Packet**Table 77 write Response**

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x05
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG RC_INVALID_STATE (AP is not ready to receive the data block because the prepare command was not sent prior to the write command) See Table 9 for response code descriptions.

read**Description**

The *read* command is used to download data from the manager's flash. The *address* field indicates the start address of the flash contents to be read. The *length* field indicates how many bytes should be read. The *length* field must be less than or equal to 120. The response contains the requested data.

Request Packet**Table 78 read Request**

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x06
Sequence	Byte	
Address	LongInt	
Length	ShortInt	

Response Packet**Table 79 read Response**

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x06
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_ARG See Table 9 for response code descriptions.
Data	Byte[]	

verify**Description**

The *verify* command is used to verify the software image indicated in the most recent *prepare* command. This command should be used only after all write commands have been issued for a particular image.

Request Packet**Table 80 verify Request**

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x07
Sequence	Byte	

Response Packet**Table 81 verify Response**

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x07
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_INVALID_STATE (the image could not be verified because a prepare command was not previously issued) See Table 9 for response code descriptions.

boot

Description

The *boot* command causes the bootloader to switch to running the LM2650 application software.

Request Packet

Table 82 boot Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x08
Sequence	Byte	

Response Packet

Table 83 boot Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x08
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK RC_CMD_FAILED See Table 9 for response code descriptions.

reset

Description

The *reset* command initiates a reset of the LM2650 manager. The current API session will be terminated.

Request Packet

Table 84 reset Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x09
Sequence	Byte	

Response Packet

Table 85 reset Response

Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x09
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK See Table 9 for response code descriptions.

shutDown

Description

The *shutDown* command places the manager in low power sleep mode to save power during commissioning. When the manager is shut down, the network is disconnected. To restart the manager when it is in low power sleep mode, perform a hardware reset.

Request Packet

Table 86 shutDown Request

Field Name	Data Type	Value/Description
Control	Byte	0x02
Type	Byte	0x0A
Sequence	Byte	

Response Packet

Table 87 shutDown Response

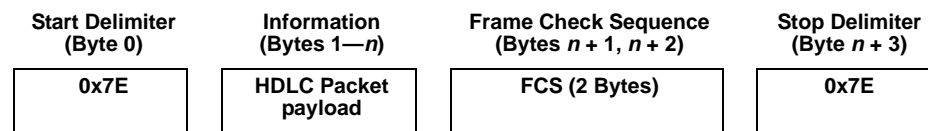
Field Name	Data Type	Value/Description
Control	Byte	0x03
Type	Byte	0x0A
Sequence	Byte	
Response code	Byte	Valid response codes: RC_OK See Table 9 for response code descriptions.

HDLC Packet Format

This appendix describes the HDLC framing used by the LM2650 manager serial protocol, and provides examples of HDLC packet processing.

HDLC Packet

The following diagram summarizes the packet format:



Start and Stop Delimiters

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

Information

The *information* field is LM2650 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* (FCS) field 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* and *stop* bits that may be added for asynchronous transmission. This specifically does not include the *start* and *stop* delimiter or the FCS field itself. The FCS field must be calculated using the algorithm specified in RFC1662.

Octet Stuffing

The LM2650 manager serial protocol uses octet stuffing to escape the *start* and *stop* delimiter (0x7E) and Control Escape (0x7D) bytes that may be contained in the *information* or *frame check sequence* 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 *frame check sequence* 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 3.

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* and *stop* flags (RFC 1662).

Note that some products may require an additional 7E start byte for high speed operation (refer to the 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 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”.

Index

A

access control list. *see* ACL
ACL 15
ACL device
 deleteACLDevice 19
 getNextACLDevice 27
 setACLDevice 36

B

bandwidth profiles 16, 26
bootloader API 2, 45
 establishing bootloader session 47
bootloader commands 48
 boot 52
 prepare 49
 read 50
 reset 52
 shutDown 53
 verify 51
 write 49
bootloader mode 2
bootup process 2
broadcast packet 35
byte ordering 4

C

clearStatistics 18
command list 17, 48
command response codes 16
control field 4

D

data types 4
deleteACLDevice 18, 19
delimiters 55

E

exchangeNetworkId 20

exchangeNetworkKey 18

F

FCS
FixedPoint data type 4
frame check sequence. *See* FCS

G

getLog 18, 20
getMote 17, 21
getMoteStatistics 18, 23
getNetStatistics 18, 25
getNetwork 17, 26
getNextACLDevice 17, 27
getNextPathStatistics 27
getPathStatistics 18
getSystem 17, 29
getTime 17, 29

H

HDLC packet format 3
HELLO packet 6, 11
HELLO_RESPONSE packet 12

I

information field 55

L

Log notification 41
LongInt data type 4

M

manager reset, detecting 8
manager time 29
MGR_HELLO packet 11

- mote
 - getMote 21
 - getMoteStatistics 23
 - pingMote 30
 - resetting 34
- mote states 23

N

- network
 - getNetStatistics 25
 - getNetwork 26
 - resetting 34
 - setNetwork 37
- network bandwidth control 16
- network event types
 - brownout 44
 - command finished 42
 - mote join 43
 - mote lost 43
 - mote operational 43
 - mote reset 42
 - mote unknown 43
 - network time 43
 - ping reply 44
- network ID 20
- notification packet 39
- notifications 2, 39

O

- octet stuffing 56
- operational mode 2

P

- packet contents
 - control field 4
 - payload field 5
 - sequence number field 5
 - type field 5
- packet field descriptions 4
- packet format 3
 - HDLC packet format 3
 - HELLO packet 11
 - HELLO_RESPONSE packet 12
 - LM2650 protocol packet format 3
 - MGR_HELLO packet 11
 - notification packet 39
- path
 - getNextPathStatistics 27
- payload field 5
- pingMote 18, 30
- profiles 16, 26

R

- radio test 31, 32
- radioTestRx 18, 31
- radioTestStatistics 18
- radioTestStats 32
- radioTestTx 18, 32
- reset 18, 34
 - detecting manager reset 8
- response codes 16

S

- sendRequest 17, 35
- sequence number field 5
- serial data notification 40
- serial session, establishing 6
- session establishment commands 10
- session packet processing 13
- setACLDevice 17, 36
- setNetwork 17, 37
- ShortInt data type 4
- SmartMesh-enabled Network 1
- software update 47
- state machine
 - manager 9
 - OEM controller 8
- statistics
 - getMoteStatistics 23
 - getNetStatistics 25
 - getNextPathStatistics 27
- subscribe 17, 38
- system
 - getSystem 29
 - resetting 34
- system event notification 41

T

- type field 5

W

- wireless communication types 14