SmartMesh IP Embedded Manager API Guide

Table of Contents

1 About This Guide

1.1 Related Documents

The following documents are available for the SmartMesh IP network:

Getting Started with a Starter Kit

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

User's Guide

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

Interfaces for Interaction with a Device

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

Software Development Tools

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

Application Notes

- SmartMesh IP Application Notes - Cover a wide range of topics specific to SmartMesh IP networks and topics that apply to SmartMesh networks in general.

Documents Useful When Starting a New Design

- The Datasheet for the LTC5800-IPM SoC, or one of the modules based on it.
- The Datasheet for the LTC5800-IPR SoC, or one of the embedded managers based on it.

- A Hardware Integration Guide for the mote/manager SoC or module - this discusses best practices for integrating the SoC or module into your design.
- A Hardware Integration Guide for the embedded manager - this discusses best practices for integrating the embedded manager into your design.
- A Board Specific Integration Guide - For SoC motes and Managers. Discusses how to set default IO configuration and crystal calibration information via a "fuse table".
- Hardware Integration Application Notes - contains an SoC design checklist, antenna selection guide, etc.
- The ESP Programmer Guide - a guide to the DC9010 Programmer Board and ESP software used to load firmware on a device.
- ESP software - used to program firmware images onto a mote or module.
- Fuse Table software - used to construct the fuse table as discussed in the Board Specific Configuration Guide.

Other Useful Documents

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

1.2 Conventions Used

The following conventions are used in this document:

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

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

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

Tips provide useful information about the product.

Informational text provides additional information for background and context

Notes provide more detailed information about concepts.

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

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

1.3 Revision History

Revision	Date	Description
1	07/18/2012	Initial Release
2	08/10/2012	Updated radiotestTx API
3	03/18/2013	Numerous Small Changes
4	10/22/2013	Corrected radioTx fields; Clarification of RC handling; Other minor changes
5	04/04/2014	Updated and clarified radiotest commands;
6	10/23/2014	Included command IDs in titles; Clarified maxMotes setting; Other minor changes
7	04/22/2015	Deprecated autostart command; Deprecated setTime command; Fixed queue occupancy description; Other minor changes
8	12/03/2015	Deprecated software licencing commands; Renamed guide to focus on embedded manager; Other minor changes
9	11/07/2016	Added joinFailed and invalidMIC events; Added RC_UNSUPPORTED response code; Added RSSI report description; Added fields to getMoteInfo; Added blink notification

2 Introduction

This guide describes the commands used to communicate with the SmartMesh IP Manager through the serial Application Programming Interface (API). The API is intended for machine-to-machine communications (e.g. a host program talking to the manager).

In contrast, the Command Line Interface (CLI) is intended for human interaction with a manager, e.g. during development, or for interactive troubleshooting. See the SmartMesh IP Manager CLI Guide for details on that interface.

2.1 Communications

Messages are sent using a packet-based protocol. For an overview of the manager capabilities referenced in this document, please see the SmartMesh IP User's Guide.

To use the Serial API, a client first establishes a session with the manager as described in Communication between Manager and Client. Once a session is established, the client can send a series of Commands to the manager to control network operations or send data to motes in the network. A client uses the *subscribe* command to request Notifications from the manager.

The Serial API can be accessed directly via the serial port using the HDLC serial protocol. Each command request, response and notification is packed as an HDLC frame. The protocol used by the Serial API is described in the Protocol section.

The Serial API describes structures for:

- Commands: Commands are RPC-style requests initiated by the client with responses returned by the Manager.
- Notifications: Notifications are asynchronous messages from the Manager to the client.

Since the manager serial connection only works with one physical client endpoint, the Serial API can also be accessed through the Serial API Multiplexer (Serial Mux) ("Serial Mux"). The Serial Mux allows multiple simultaneous clients to connect to the Manager, e.g. a user GUI and a statistics logger. The underlying Command and Notification structures are the same, but Serial Mux uses a different packet format for encalsulating messages. The Serial Mux documentation describes its message format.

3 Protocol

The SmartMesh IP Manager Serial API protocol is a packet-based communication protocol used to enable communications over an asynchronous serial port. By default, the protocol runs over RS232 at 115kbps, 8 bits, no parity, 1 stop bit. RTS/CTS hardware handshaking is used, and is not covered in this document.

- The Packet Format section describes the HDLC format of packets sent by each side.
- The Data Representation section describes how data types are serialized.
- The Communication between Manager and Client section describes how communication is initiated between the Manager and client.

3.1 Data Representation

3.1.1 Common Data Types

The following data types are used in this API guide for data representation.

Type	Length (bytes)	Notes
INT8U	1	Unsigned byte.
INT16U	2	Short unsigned integer.
INT32U	4	Long unsigned integer.
INT8S	1	Signed byte or character.
INT16S	2	Short signed integer.
INT32S	4	Long signed integer.
INT8U[n]	n	Fixed size array. Fixed size arrays always contain [n] elements. Fixed size arrays that contain fewer valid values are padded to the full length with a default value.
INT8U[]	variable	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.
IPV6_ADDR	16	IPV6 address, represented as INT8U[16] byte array.
ASN	5	Absolute slot number (ASN) is the number of timeslots since network startup, represented as a 5 byte integer.

UTC_TIME	8	UTC Time is the number of seconds and microseconds since midnight January 1, 1970 UTC. The serialized format is as follows: • INT32U - seconds - number of seconds since midnight of January 1, 1970. • INT32U - microseconds - microseconds since the beginning of the current second.
UTC_TIME_L	12	Long UTC Time is the number of seconds and microseconds since midnight January 1, 1970 UTC. The serialized format is as follows: • INT64 - seconds - number of seconds since midnight of January 1, 1970. • INT32 - microseconds - microseconds since the beginning of the current second.
MAC_ADDR	8	EUI-64 identifier, or MAC address, represented as INT8U[8] byte array.
SEC_KEY	16	Security key, represented as INT8U[16] byte array.
BOOL	1	True(=1), False(=0). A boolean field occupies a full byte.
APP_VER	5	Application version. The serialized format is as follows: • INT8U - major - the major version • INT8U - minor - the minor version • INT8U - patch - the patch version • INT16U - build - the build version

3.1.2 Integer Representation

All multi-byte numerical fields are represented as octet strings in most-significant-octet first order. All octets are represented as binary strings in most-significant-bit first order. Signed integers are represented in two's complement format.

INT8S, INT8U

Bit 7	...	Bit 0
MSB		LSB

INT16S, INT16U

Bit 15 ... Bit 8	Bit 7 ... Bit 0

INT32S, INT32U

Bit 31 ... Bit 24	Bit 23 ... Bit 16	Bit 15 ... Bit 8	Bit 7 ... Bit 0

ASN

Bit 39 ... Bit 32	Bit 31 ... Bit 24	Bit 23 ... Bit 16	Bit 15 ... Bit 8	Bit 7 ... Bit 0

3.1.3 Transmission Order

All structures in this document are depicted in the order in which they are transmitted - from left to right (or, in the case of tables, top to bottom).

3.2 Packet Format

3.2.1 Framing

Each packet is an HDLC frame as outlined in RFC 1662 delimited by flags with a 2 byte Frame Check Sequence (FCS).

Flag	Data	Frame Check Sequence	Flag
1 byte	N bytes	2 bytes	1 byte

Flags

The protocol uses `0x7E` (the “flag sequence”) for packet delineation. Every packet must start and end with this flag. Instances of the flag sequence must be re-encoded to avoid framing errors. This “octet-stuffing” procedure is described in RFC 1662.

Frame Check Sequence (FCS)

The 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/stop bits that may be added for asynchronous transmission. This specifically does not include the Flag sequences or the FCS field itself. The field is calculated per the algorithm specified in RFC 1662.

3.2.2 Data Payload

All packets in the contain the following fields in the Data section:

Control	Packet Type	Seq. Number	Payload Length	Payload
1 byte	1 byte	1 byte	1 byte	0-N bytes (type-specific)

The Control, Packet Type, Sequence Number and Length fields are the Serial API Header.

Control

Control is a mandatory bitmap field contained in the beginning of every packet. At present, the following fields are defined:

Bit 0 (DATA/ACK)	0 = data packet (i.e. non-ack) 1 = ack packet
Bit 1 (SERVICE_TYPE)	0 = unacknowledged 1 = acknowledged
Bits 2-7	Reserved, set to 0

Bit 0 indicates whether the packet is a data packet or an ack packet. Requests are *data* packets and responses are *ack* packets.

Bit 1 indicates whether the packet should be acknowledged or not. The session establishment handshake packets (*hello*, *mgrHello* and *helloResponse*) are unacknowledged (no ack is required).

Packet Type

Packet Type is a mandatory field indicating the structure of the packet payload that follows. The Packet Type values for each message are listed in the Packet and Command Types table. For ack-only packets containing no application payload, Packet Type should be set to the same value as the packet that is being acknowledged.

Sequence Number

Sequence Number is a field used for reliable data communication only. This field is ignored for packets marked as unacknowledged. In regular (non-ack) packets, it contains the sender's unique reliable packet number. In ack packets, this field contains the sequence number of packet being acknowledged.

Payload Length

Payload Length is the size in bytes of the payload part of the packet. The maximum packet size originated and terminated by the Manager is 128 bytes, which includes the API header and payload, but *not* HDLC flags, escapes, or the frame checksum.

Payload

The presence and contents of the Payload is determined by the Type field. A Payload can be present in both regular (non-ack) and ack packets.

The structures of Commands and Notifications are described in their respective sections.

3.3 Communication Between Manager and Client

A client communicates with the Manager Serial API over the manager's API serial port. This section describes how a client initiates a session with the manager.

3.3.1 Initiating Communication

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. Once the session is established, the session continues until one of the following occurs:

- the manager resets,
- the client does not acknowledge a notification, or
- the client sends new *hello* packet.

MgrHello Packet

The *mgrHello* packet is sent by the manager to indicate that it is ready to initiate a new session with the client. The *mgrHello* packet also indicates that the manager has disconnected an existing session with a client. When the manager has no established client session, it periodically sends *mgrHello* packets. The payload of the *mgrHello* packet is as follows:

Parameter	Type	Description
version	INT8U	Version of the protocol. This document describes protocol version 4.
mode	INT8U	Reserved for compatibility; must be 0

Hello Packet

The *hello* packet is sent by the client to initiate new session with the manager. The payload of the *hello* packet is as follows:

Parameter	Type	Description
version	INT8U	Version of the Protocol supported by the client. The manager checks this field to decide on compatibility with the client. If the protocol version is not supported by the manager, the *helloResponse* will contain an unsupported version error code, and the version field will contain the supported version. This document describes protocol version 4.
cliSeqNo	INT8U	The client sequence number is the unique number of this *hello* packet. Used for reliable communication, both sides use unique numbers to detect duplicate reliable messages. Once the *helloResponse* is received, the client's reliable commands must start with the "next" sequence number.
mode	INT8U	Reserved for compatibility; must be 0

HelloResponse Packet

The *helloResponse* packet is sent by the manager to the client in response to a client's Hello. If the *responseCode*is OK, the Manager and client have established a session and can begin exchanging commands. Otherwise, the client should retry. The payload of the *helloResponse* packet is as follows:

Parameter	Type	Description
responseCode	INT8U	Response code is used by the manager to indicate the result of session establishment. The response code values for this command are defined below.
version	INT8U	Protocol version supported by this Manager. If OK is returned in *responseCode*, the protocol version should match that in the *hello* message.
mgrSeqNo	INT8U	The manager sequence number establishes the sequence number of manager's reliable data stream. Subsequent reliable messages sent by the manager will use the next sequence number.
cliSeqNo	INT8U	Confirms the client's reliable sequence number received in *hello* message. The client must send reliable commands starting with the "next" sequence number.
mode	INT8U	Reserved for compatibility; must be 0

Response Codes

Response code	Value	Description
OK	0	Session established
unsupportedVersion	1	The client's protocol version is not supported
invalidMode	2	The manager is running in a mode that does not support the Serial API

3.3.2 Reliable (Acknowledged) Communication

Once the manager and client have established a session, all request-response communication is reliable, i.e. all request packets must be explicitly acknowledged by the receiver. The response packet contains a link-layer acknowledgement and can contain an optional application-layer response (depending on the Type). The application layer message included in the response is not acknowledged separately.

The sender should not send the next packet until the previous packet is acknowledged. The sender stores the sequence number of each outgoing packet. If a request packet is not acknowledged within a small time window, the sender should resend the packet with the same sequence number. After several retries, the sender should consider the session disconnected and should reestablish the session with the *hello* exchange.

There are few requirements on how sequence numbers should be managed by the receiver:

1. The receiver stores the sequence number of each (acknowledged) packet. The incoming sequence number is stored separately from the outgoing sequence number.
2. The receiver processes an incoming packet if its sequence number is different from the previous (stored) sequence number.
3. If the sequence number of the incoming packet is the same, the receiver acknowledges the duplicate packet without processing it.
4. 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.

The manager will try sending each acknowledged packet up to 3 times, with 200ms delay in between. If no valid reply is received, the manager will terminate the connection and drop the pending packet. After the connection is dropped, new client must complete a new *hello* packet exchange.

3.3.3 Best-Effort (Unacknowledged) Communication

When client subscribes to Manager's notifications, it can optionally request that the manager sent selected notifications unacknowledged (i.e. best effort). This may be useful to improve overall packet throughput for data traffic coming from the network. Packets sent best-effort will be marked as 'unacknowleged' by the Manager (see Control field). Such packets do not require the client to send back an explicit reply.

3.3.4 Guidelines for Writing Forward-Compatible Clients

The serial API protocol is designed to allow clients to remain compatible with newer releases of software. The following changes should be expected to occur with future revisions of software:

- Payload structures may be extended to include new fields. New fields will either be added at the end or in place of reserved bytes.
- Fields may be marked as deprecated, but will not be removed from payload structures.
- New commands and notifications may be added.
- New alarms and events may be added.
- New response codes may be added.

To remain compatible, a client should observe the following rules:

- If the client receives response payload that is longer than expected, it should silently ignore the extra bytes and process the known bytes only.
- If the client receives a packet with unrecognized notification type, it should acknowledge it with RC_OK.
- If the client receives an unrecognized alarm or event it should acknowledge the notification with RC_OK.
- Never rely on value of reserved fields – they should be ignored.
- If a field is marked as unused or reserved in request payload, its value should be set to zeros, unless otherwise noted.
- If an unrecognized response code is received, it should be treated as an general error response code.

If the protocol changes in any other incompatible way, the protocol version will be changed.

4 Commands

Once a client has established a session, it can send commands to the manager. Commands are always sent from the client to the manager. Commands are always reliable. Commands are acknowledged by the manager with the response.

The Serial API Header (see Packet Format) contains the *packet type*, which allows the receiver to identify the structure contained in the payload. The command and notification structures defined in the Serial API are also used by the Serial Mux API. The Serial Mux uses a different header structure, but the same request and response structure.

The request structure is serialized in the payload section of a serial API packet. A request with no parameters is an empty payload.

Serial API Header	Request structure, 0-N bytes, (see specific command description)

The packet type for responses is the same as the packet type for the request. Responses always contain a response code (RC) in the first byte of payload followed by the rest of the serialized response structure.

Serial API Header	Response structure, 1-N bytes, (see specific command description)

All commands will return RC_OK (0) if the command succeeds, and a nonzero RC if the command fails. Command descriptions list currently defined nonzero RCs. While the content of the RC provides some information on the nature of the failure, a forward-compatible client should be prepared to accept other nonzero response codes not explicitly listed for the command.

Each command is assigned a unique *packet type* value, which is part of the Serial API Header. The values are listed in the Packet and Command Types table.

The following commands are defined:

4.1 clearStatistics (0x1F)

Description

The *clearStatistics* command clears the accumulated network statistics. The command does not clear path quality or mote statistics.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed

4.2 deleteACLEntry (0x29)

Description

The *deleteACLEntry* command deletes the specified mote from the access control list (ACL). If the *macAddress* parameter is set to all 0xFFs or all 0x00s, the entire ACL is cleared. This change is persistent.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	Specified mote is not found in ACL
RC_WRITE_FAIL	Flash write error, can't save new settings

4.3 deleteMote (0x45)

Description

The *deleteMote* command deletes a mote from the manager's list. A mote can only be deleted if it in the *Lost* or *Unknown* states. This change is persistent.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	Specified mote is not found
RC_INV_STATE	Mote state is not Lost or mote is access point
RC_WRITE_FAIL	Flash write error, can't save new settings

4.4 exchangeMoteJoinKey (0x21)

Description

The *exchangeMoteJoinKey* command triggers the manager to send a new join key to the specified mote and update the manager's ACL entry for the mote. The response contains a *callbackId*. A commandFinished event notification with this *callbackId* will be sent when the operation is complete. This change is persistent.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address
key	SEC_KEY		New mote join key

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
callbackId	INT32U		Callback ID

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	Mote with specified MAC address is not found
RC_INV_STATE	Mote is not in operational state
RC_NACK	User commands queue is full
RC_WRITE_FAIL	Flash write error, can't save new settings

4.5 exchangeNetworkId (0x22)

Description

The *exchangeNetworkId* command triggers the manager to distribute a new network ID to all the motes in the network. A *callbackId* is returned in the response. A commandFinished notification with this *callbackId* will be sent when the operation is complete. This change is persistent.

Request

Parameter	Type	Enum	Description
id	INT16U		Network ID

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
callbackId	INT32U		Callback ID

Response Codes

Code	Description
RC_OK	Command received
RC_NACK	User commands queue is full
RC_IN_PROGRESS	A command is still pending. Wait until a commandFinished notification is received for the previous command before retrying.
RC_WRITE_FAIL	Flash write error; cannot save new settings

4.6 getIPConfig (0x43)

Description

The *getIPConfig* command returns the manager's IP configuration parameters, including the IPv6 address and mask.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
ipv6Address	IPV6_ADDR		IPv6 address
mask	INT8U[16]		Subnet mask

Response Codes

Code	Description
RC_OK	Command successfully completed

4.7 getLicense (0x37)

Description

The *getLicense* command has been deprecated in Manager >= 1.3.0 .There is no need to use a license to enable > 32 mote networks.

The *getLicense* command returns the current license key.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
license	INT8U[13]		Current software license key

Response Codes

Code	Description
RC_OK	Command successfully completed

4.8 getLog (0x2B)

Description

The *getLog* command retrieves diagnostic logs from the manager or a mote specified by MAC address.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	Specified mote not found
RC_INV_STATE	Mote is not in operational state

4.9 getManagerStatistics (0x35)

Description

The *getManagerStatistics* command returns dynamic information and statistics about the manager API. The statistics counts are cleared together with all current statistics using *clearStatistics*.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
serTxCnt	INT16U		Number of packets sent out on the serial port. This value may roll over if not cleared.
serRxCnt	INT16U		Number of packets received on the serial port. This value may roll over if not cleared.
serRxCRCErr	INT16U		Number of CRC errors
serRxOverruns	INT16U		Number of overruns detected
apiEstabConn	INT16U		Number of established Serial API connections
apiDroppedConn	INT16U		Number of dropped Serial API connections
apiTxOk	INT16U		Number of request packets sent on serial API for which ack-OK was received
apiTxErr	INT16U		Number of request packets sent on serial api for which acknowledgment error was received
apiTxFail	INT16U		Number of packets for which there was no acknowledgment
apiRxOk	INT16U		Number of request packets that were received and acknowledged
apiRxProtErr	INT16U		Number of packets that were received and dropped due to invalid packet format

Response Codes

Code	Description
RC_OK	Command successfully completed

4.10 getMoteConfig (0x2F)

Description

The *getMoteConfig* command returns a single mote description as the response. The command takes two arguments, a MAC Address and a flag indicating whether the MAC Address refers to the requested mote or to the next mote in manager's memory. This command may be used to iterate through all motes known by the manager by starting with the *macAddress* parameter set to 0 and *next* set to true, and then using the MAC Address of that response as the input to the next call.

The mote MAC address is used in all query commands, but space constraints require the neighbor health reports to use the Mote ID for identification. Therefore, both identifiers are present in the mote structure.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address
next	BOOL		True if looking for next mote; false if looking for this MAC address

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
macAddress	MAC_ADDR		Mote MAC address
moteId	INT16U		Mote ID (used in health reports)
isAP	BOOL		Indicates that this is the Manager access point
state	INT8U	Mote State	Mote state
reserved	INT8U		Reserved, values should be ignored.
isRouting	BOOL		Indicates whether this mote can be used as a non-leaf node in the network

Response Codes

Code	Description
RC_OK	Command successfully completed

RC_NOT_FOUND	The specified mote doesn't exist
RC_END_OF_LIST	Last mote in the list has been reached (next = true)

4.11 getMoteConfigById (0x41)

Description

The *getMoteConfigById* command returns a single mote description as the response. The command takes one argument, the short address of a mote (Mote ID). The command returns the same response structure as the *getMoteConfig* command.

Request

Parameter	Type	Enum	Description
moteId	INT16U		Mote ID

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
macAddress	MAC_ADDR		Mote MAC address
moteId	INT16U		Mote ID (used in health reports)
isAP	BOOL		Indicates this is the Manager access point
state	INT8U	Mote State	Mote state
reserved	INT8U		Reserved, values should be ignored.
isRouting	BOOL		Indicates whether this mote can be used as a non-leaf node in the network

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	No such mote

4.12 getMoteInfo (0x3E)

Description

The *getMoteInfo* command returns dynamic information for the specified mote.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
macAddress	MAC_ADDR		Mote MAC address
state	INT8U	Mote State	Mote state
numNbrs	INT8U		The number of motes within range of this mote, both currently and potentially connected
numGoodNbrs	INT8U		The number of neighboring motes that have good (> 50) quality paths with this mote
requestedBw	INT32U		Bandwidth requested by mote, milliseconds per packet
totalNeededBw	INT32U		Total bandwidth required by the mote and its children (includes requestedBw), milliseconds per packet
assignedBw	INT32U		Currently assigned bandwidth, milliseconds per packet
packetsReceived	INT32U		Number of packets received by the manager from the mote
packetsLost	INT32U		Number of packets sent by the mote, but lost at the manager. The number of packets lost is calculated using the security counter, so it trails real-time by the size of the packet window (a lost packet is not recorded until WINDOW_SIZE more packets are transmitted).
avgLatency	INT32U		

			The average time (in milliseconds) taken for packets generated at the mote to reach the manager
stateTime	INT32U		Time after last mote state modification (sec). (Added in Manager 1.3.0)
numJoins	INT8U		Number of times this device has joined. This field saturates at 255 joins. (Added in Manager 1.4.1)
hopDepth	INT8U		Calculated number of hops from the manager times 10. If there are more than 25 hops, this field does not increase beyond 255. (Added in Manager 1.4.1)

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	No such mote

4.13 getMoteLinks (0x46)

Description

The *getMoteLinks* command returns information about links assigned to the mote. The response contains a list of links starting with Nth link on the mote, where N is supplied as the *idx* parameter in the request. To retrieve all links on the device the user can call this command with *idx* that increments by number of links returned with prior response, until the command returns RC_END_OF_LIST response code. Note that links assigned to a mote may change between API calls.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mac address of the mote.
idx	INT16U		Starting index of the links to return.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
idx	INT16U		Starting index of the first link returned (0-based).
utilization	INT8U		Bandwidth utilization of all links on this mote (0-31).
numLinks	INT8U		Number of links in this response.
links	links[]		Array of 'numLinks' links, see structure below.

Link structure returned in the response

Parameter	Type	Enum	Description
frameId	INT8U		Frame id
slot	INT32U		Slot number
channelOffset	INT8U		Channel Offset
moteId	INT16U		Peer mote id
flags	INT8U	Link Flags	Link flags

Response Codes

Code	Description
RC_NOT_FOUND	No such mote.
RC_INV_STATE	Mote is not in operational state
RC_END_OF_LIST	The index requested is greater than number of links.

4.14 getNetworkConfig (0x3F)

Description

The *getNetworkConfig* command returns general network configuration parameters, including the Network ID, bandwidth parameters and number of motes.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
networkId	INT16U	none	Network ID
apTxPower	INT8S	Transmit Power	Access Point transmit power
frameProfile	INT8U	Frame Profile	The frame profile describes the length of the slotframes during network building and normal operation. All of the legacy frame profiles are mapped to the standard IP manager profile: Profile 1 (fast build, medium speed operation).
maxMotes	INT16U	none	The maximum number of motes allowed in the network.
baseBandwidth	INT16U	none	Base bandwidth is the default bandwidth allocated to each mote that joins.
downFrameMultVal	INT8U	none	Downstream frame multiplier is a multiplier for the length of the primary downstream slotframe.
numParents	INT8U	none	Number of parents allocated to each mote.
ccaMode	INT8U	CCA Mode	Indicates the mode for Clear Channel Assessment (CCA) in the network.
channelList	INT16U	none	Bitmap of channels in the whitelist to use for communication, all others blacklisted. Bit 0×0001 corresponds to channel 0 and bit 0×8000 corresponds to channel 15. (0=not used, 1=used).
autoStartNetwork	BOOL	none	The Auto Start Network flag tells the Manager whether to start the network as soon as the device is booted. (deprecated)

locMode	INT8U	none	Reserved
bbMode	INT8U	Backbone Frame Mode	Backbone frame mode
bbSize	INT8U	none	Backbone frame size
isRadioTest	INT8U	none	Indicates whether the Manager is in radiotest mode
bwMult	INT16U	none	Bandwidth provisioning multiplier in percent (100-1000)
oneChannel	INT8U	none	Channel number for One Channel mode. 0xFF = One Channel mode is OFF

Response Codes

Code	Description
RC_OK	Command successfully completed

4.15 getNetworkInfo (0x40)

Description

The *getNetworkInfo* command returns dynamic network information and statistics.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Description
numMotes	INT16U		Number of motes in the "Operational" state (not including the Access Point)
asnSize	INT16U		ASN size is the timeslot duration, in microseconds
advertisementState	INT8U	Advertisement State	Advertisement state
downFrameState	INT8U	Downstream Frame Mode	Indicates the current downstream frame length, that is, whether or not the multiplier is applied
netReliability	INT8U		Network reliability as a percentage
netPathStability	INT8U		Path stability as a percentage
netLatency	INT32U		Average latency, in milliseconds
netState	INT8U	Network State	Current network state
ipv6Address	IPV6_ADDR		IPV6 address of the system
numLostPackets	INT32U		Number of lost packets (Added in Manager 1.3.0)
numArrivedPackets	INT64U		Number of received packets (Added in Manager 1.3.0)
maxNumbHops	INT8U		Maximum number of hops in the network * 10 (Added in Manager 1.3.0). This is the largest value across all motes for average hops traveled by a mote's upstream packets.

Response Codes

Code	Description
RC_OK	Command successfully completed

4.16 getNextACLEntry (0x28)

Description

The *getNextACLEntry* command returns information about next mote entry in the access control list (ACL). To begin a search (find the first mote in ACL), a zero MAC address (0000000000000000) should be sent.

⚠ There is no mechanism for reading the ACL entry of a specific mote. This call is an iterator. If you call *getNextACLEntry* with mote A as the argument, your response is the ACL entry for mote B, where B is the next mote in the ACL.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
macAddress	MAC_ADDR		Mote MAC address
joinKey	SEC_KEY		No join key reads are permitted, returns 0s

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_END_OF_LIST	End of ACL is reached
RC_NOT_FOUND	No such mote in the ACL

4.17 getNextPathInfo (0x31)

Description

The *getNextPathInfo* command allows iteration across paths connected to a particular mote. The *pathId* parameter indicates the previous value in the iteration. Setting *pathId* to 0 returns the first path. A *pathId* can not be used as a unique identifier for a path. It is only valid when associated with a particular mote.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address
filter	INT8U	Path Filter	Specifies whether the command iterates through all paths connected to the mote, or only the upstream paths
pathId	INT16U		Path ID

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
pathId	INT16U		Path ID
source	MAC_ADDR		MAC address of the source mote
dest	MAC_ADDR		MAC address of the destination mote
direction	INT8U	Path Direction	Path direction
numLinks	INT8U		Number of links
quality	INT8U		An internal measurement of path quality based on a moving average of packets received over packets transmitted. There are no meaningful units for this value. Highest values are better. Range 0 (worst) to 100 (best). Quality is equivalent to path stability for a used path, and is based on RSSI for an unused path.
rssiSrcDest	INT8S		Latest RSSI or 0 (if there is no data), for the path from source mote to destination mote. Calculated only for paths after the first health report.

rssiDestSrc	INT8S		Latest RSSI or 0 (if there is no data), for the path from destination mote to source mote. Calculated only for paths after the first health report.

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	The specified path ID does not exist
RC_END_OF_LIST	The specified pathId in the request is the end of the list

4.18 getPathInfo (0x30)

Description

The *getPathInfo* command returns parameters of requested path.

Request

Parameter	Type	Enum	Description
source	MAC_ADDR		MAC address of source mote
dest	MAC_ADDR		MAC address of destination mote

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
source	MAC_ADDR		MAC address of source mote
dest	MAC_ADDR		MAC address of destination mote
direction	INT8U	Path Direction	Path direction
numLinks	INT8U		Number of links between motes for upstream frame
quality	INT8U		An internal measurement of path quality based on a moving average of packets received over packets transmitted. There are no meaningful units for this value. Highest values are better. Range 0 (worst) to 100 (best). Quality is equivalent to path stability for a used path, and is based on RSSI for an unused path.
rssiSrcDest	INT8S		Latest RSSI or 0 (if there is no data), for the path from source mote to destination mote. Calculated only for paths after the first health report.
rssiDestSrc	INT8S		Latest RSSI or 0 (if there is no data), for the path from destination mote to source mote. Calculated only for paths after the first health report.

Response Codes

Code	Description

RC_OK	Command successfully completed
RC_NOT_FOUND	A path between the specified motes doesn't exist

4.19 getRadiotestStatistics (0x26)

Description

This command retrieves statistics from a previously run *radiotestRx* command. It may only be executed if the manager has been booted up in radiotest mode (see *setNetworkConfig* command).

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
rxOk	INT16U		Number of packets received successfully
rxFail	INT16U		Number of packets received with errors

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_IN_PROGRESS	Radiotest is in progress
RC_INVALID_COMMAND	No radiotest was started

4.20 getSystemInfo (0x2E)

Description

The *getSystemInfo* command returns system-level information about the hardware and software versions.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
macAddress	MAC_ADDR		MAC address
hwModel	INT8U		Hardware model
hwRev	INT8U		Hardware revision
swMajor	INT8U		Software version, major
swMinor	INT8U		Software version, minor
swPatch	INT8U		Software version, patch
swBuild	INT16U		Software version, build

Response Codes

Code	Description
RC_OK	Command successfully completed

4.21 getTime (0x17)

Description

The *getTime* command returns the current manager UTC time and current absolute slot number (ASN). The time values returned by this command are delayed by queuing and transfer time over the serial connection. For additional precision, an external application should trigger the networkTime notification using the Time Pin.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
uptime	INT32U		Time (sec) that the response was generated (as uptime)
utc	UTC_TIME_L		Time that the response was generated (as UTC)
asn	ASN		Absolute slot number (ASN)
asnOffset	INT16U		Offset inside ASN, in microseconds

Response Codes

Code	Description
RC_OK	Command successfully completed

4.22 pingMote (0x2A)

Description

The *pingMote* command sends a ping (echo request) to the mote specified by MAC address. A unique *callbackId* is generated and returned with the response. When the response is received from the mote, the manager generates a pingResponse notification with the measured round trip delay and several other parameters. The request is sent using unacknowledged transport, so the mote is not guaranteed to receive the request.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		MAC address of the mote

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
callbackId	INT32U		Callback ID

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	Specified mote not found
RC_INV_STATE	Mote is not in operational state
RC_NO_RESOURCES	User commands queue is full
RC_IN_PROGRESS	Previous echo request command is still pending for specified mote

4.23 radiotestRx (0x25)

Description

The *radiotestRx* command clears all previously collected statistics and initiates radio reception on the specified channel. It may only be executed if the manager has been booted up in radiotest mode (see *setNetworkConfig* command). During the test, the device keeps statistics about the number of packets received (with and without error). The test results may be retrieved using the *getRadiotestStatistics* command.

The station ID is a user selectable value. It must be set to match the station ID used by the transmitter. Station ID is used to isolate traffic if multiple tests are running in the same radio space.

Channel numbering is 0-15, corresponding to IEEE 2.4 GHz channels 11-26.

Request

Parameter	Type	Enum	Description
mask	INT16U		Mask of RF channel to use for the test. Only one channel must be selected
duration	INT16U		Duration of test (in seconds)
stationId	INT8U		Unique (0-255) station ID of this device. Must match station ID on the sender. To ignore station id of the sender, use a value of 0.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_IN_PROGRESS	Radiotest is in progress
RC_INVALID_ARGUMENT	Invalid mask value

4.24 radiotestTx (0x23)

Description

The *radiotestTx* command allows the user to initiate a radio transmission test. It may only be executed if the manager has been booted up in radiotest mode (see *setNetworkConfig* command). Four types of transmission tests are supported:

- Packet transmission
- Continuous modulation (CM)
- Continuous wave, i.e unmodulated signal (CW)
- Packet transmission with clear channel assessment (CCA) enabled (Available in Manager > 1.3.x)

In a packet transmission test, the device generates a *repeatCnt* number of packet sequences. Each sequence consists of up to 10 packets with configurable size and delays. Each packet starts with a PHY preamble (5 bytes), followed by a PHY length field (1 byte), followed by data payload of up to 125 bytes, and finally a 2-byte 802.15.4 CRC at the end. Byte 0 of the payload contains stationId of the sender. Bytes 1 and 2 contain the packet number (in big-endian format) that increments with every packet transmitted. Bytes 3..N contain a counter (from 0..N-2) that increments with every byte inside payload. Transmissions occur on the set of channels defined by *chanMask*, selected in pseudo-random order.

In a continuous modulation test, the device generates continuous pseudo-random modulated signal, centered at the specified channel. The test is stopped by resetting the device.

In a continuous wave test, the device generates an unmodulated tone, centered at the specified channel. The test tone is stopped by resetting the device.

In a packet transmission with CCA test, the device is configured identically to that in the packet transmission test, however the device does a clear channel assessment before each transmission and aborts that packet if the channel is busy.

Channel numbering is 0-15, corresponding to IEEE 2.4 GHz channels 11-26.

The station ID is a user selectable value. It is used in packet tests so that a receiver (see *radiotestRx*) can identify packets from this device in cases where there may be multiple tests running in the same radio space. This field is not used for CM or CW tests. (Available in Manager >= 1.3.0)

Request

Parameter	Type	Enum	Description
testType	INT8U	Radio Test Types	Type of transmission test

chanMask	INT16U		Mask of channels (0–15) enabled for the test. Bit 0 corresponds to channel 0. For continuous wave and continuous modulation tests, only one channel should be enabled.
repeatCnt	INT16U		Number of times to repeat the packet sequence (0=do not stop). Applies only to packet transmission tests.
txPower	INT8S		Transmit power, in dB. Valid values are 0 and 8.
seqSize	INT8U		Number of packets in each sequence. This parameter is only used for packet tests
sequenceDef	seqDef[]		Array of *seqSize* sequence definitions (up to 10) specifies the length and after-packet delay for each packets. This parameter is only used for packet tests. Each sequence definition is formatted as follows: INT8U pkLen; /* Length of packet (2-125 bytes) */ INT16U delay; /* Delay after packet transmission, microseconds */
stationId	INT8U		Unique (1-255) identifier included in packets that identifies the sender. This parameter is only used for packet tests. (Available in mote >= 1.1.0)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_IN_PROGRESS	Radiotest is in progress
RC_INVALID_ARGUMENT	Invalid "channel" or "txPower" value

4.25 reset (0x15)

Description

The *reset* command is used to reset various objects. The command argument is an object type, and if the object is a mote the MAC address must be specified (otherwise that argument is ignored).

Request

Parameter	Type	Enum	Description
type	INT8U	Reset Type	Type of object to reset
macAddress	MAC_ADDR		Mote MAC address

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
macAddress	MAC_ADDR		Mote MAC address

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	Mote with specified MAC address is not found
RC_INV_STATE	Mote is not in operational state
RC_NACK	User commands queue is full (applies to mote reset)
RC_INVALID_ARGUMENT	Invalid reset type value

4.26 restoreFactoryDefaults (0x3D)

Description

The *restoreFactoryDefaults* command restores the default configuration and clears the ACL. This change is persistent.

For Manager versions <1.3.0 that required a license, the license used to enable optional features is preserved during a restore.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_WRITE_FAIL	Flash write error; cannot save new settings

4.27 sendData (0x2C)

Description

The *sendData* command sends a packet to a mote in the network. The response contains a *callbackId*. When the manager injects the packet into the network, it will generate a packetSent notification. It is the responsibility of the customer's application layer at the mote to send a response. It is also the responsibility of the customer's application layer to timeout if no response is received at the manager if one is expected.

The *sendData* command should be used by applications that communicate directly with the manager. If end-to-end (application to mote) IP connectivity is required, the application should use the *sendIP* command. For a more comprehensive discussion of the distinction, see the SmartMesh IP Network User Guide.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		MAC address of the destination mote. 0xFFFFFFFFFFFFFFFF can be used to broadcast to all motes.
priority	INT8U	Packet Priority	Priority of the packet
srcPort	INT16U		Source port
dstPort	INT16U		Destination port. This is used to route the packet to particular services on the mote.
options	INT8U		The options field is reserved for future use. It must be set to 0.
data	INT8U[]		The payload data of the packet. The data length is calculated from the overall message length.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
callbackId	INT32U		Callback ID

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NOT_FOUND	Specified mote is not found
RC_INV_STATE	Mote is not in operational state
RC_NACK	User commands queue is full or couldn't allocate memory buffer for payload
RC_INVALID_ARGUMENT	Payload size exceeds maximum allowed value

Payload Size Limits

Src/Dst Ports	Payload Size (bytes)
F0Bx	82
Any other	79

4.28 sendIP (0x3B)

Description

The *sendIP* command sends a 6LoWPAN packet to a mote in the network. The response contains a *callbackId*. When the manager injects the packet into the network, it will generate a packetSent notification with the *calllbackId*. The application is responsible for constructing a valid 6LoWPAN packet. The packet is sent to the mote best-effort, so the application should deal with responses and timeouts, if any.

The *sendIP* command should be used by applications that require end-to-end IP connectivity. For applications that do not require end-to-end IP connectivity, the *sendData* command provides a simpler interface without requiring the application to understand 6LoWPAN encapsulation. For a more comprehensive discussion of the distinction, see the SmartMesh IP Network User Guide.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		The MAC address of the destination mote. 0xFFFFFFFFFFFFFFFF can be used to broadcast to all motes.
priority	INT8U	Packet Priority	Priority of the packet
options	INT8U		Reserved for future use. The options field must be set to 0.
encryptedOffset	INT8U		Offset encrypted part of data. 0xFF - data is not encrypted
data	INT8U[]		The complete 6LoWPAN packet. The length of data field is calculated from the overall command length.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
callbackId	INT32U		Callback ID

Response Codes

Code	Description

RC_OK	Command successfully completed
RC_NOT_FOUND	Specified mote is not found
RC_INV_STATE	Mote is not in operational state
RC_NACK	User commands queue is full or could not allocate memory buffer for payload
RC_INVALID_ARGUMENT	Payload size exceeds maximum allowed value or the 6LoWPAN packet is invalid

Payload Size Limits

Src/Dst Ports	Payload size (bytes)*
F0Bx	86
F0xx	84
Any other	83

*5 bytes are occupied by the 6LoWPAN header that the application must supply

6LoWPAN Packet Format

Section	Description	Length (Bytes)	Encoding
6LoWPAN IP Header	LOWPAN_IPHC	2	011.11.1.10:C.S.ss.M.D.dd • 011 - indicates LOWPAN header • 11 - Traffic Class and Flow Label are elided • 1 - Next Header field is compressed • 10 - Hop Limit is compressed (limit 64) • C context identifier (0 = elided for current versions of LBR) • S/D source/destination compressed (1) • s/d source/destination mode: 00 – full (16B - only used for external source), 11 – elided / compressed (only for destination if M=1) (0/1B) • M multicast (0)
	Context ID Extension	0/1	SSSS.DDDD • S, D – source / destination
		0/16	

	Source Address		elided if a mesh source (e.g. manager), 16 bytes if source is external to the mesh
	Destination Address	0/1/16	elided if a mesh destination (e.g. a mote), 1 byte if multicast (not used in current version of LBR)
UDP	Header	0/1	111101.S.D • S/D– source/dest. port is compressed (1)
	Ports	1/3/4	If SD in UDP header is 11, both ports are in range F0Bx and ports are encoded as 1 byte, ssss.dddd • s, d – source / destination If SD = 10, source port is in range F0xx, and encoded as 1 byte. 2-byte destination port is carried inline If SD = 01, destination port is in range F0xx and encoded as 1 byte. 2-byte source port is carried inline If SD = 00, each 2-byte port is carried inline
Payload	User payload	Variable	

4.29 setACLEntry (0x27)

Description

The *setACLEntry* command adds a new entry or updates an existing entry in the Access Control List (ACL). This change is persistent. The maximum number of entries is 1,200.

Request

Parameter	Type	Enum	Description
macAddress	MAC_ADDR		Mote MAC address
joinKey	SEC_KEY		Join key

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_NO_RESOURCES	ACL is full (when adding a new entry)
RC_WRITE_FAIL	Flash write error, can't save new settings

4.30 setAdvertising (0x32)

Description

The *setAdvertising* command tells the manager to activate, deactivate, or use slow advertising. The response is a *callbackId*. A commandFinished notification with the *callbackId* is generated when the command propagation is complete.

With motes prior to version 1.4.1, it is only possible to turn advertising ON or OFF. If building networks consisting primarily of motes 1.4.1 or later, power can be saved by setting advertising to "slow". Set the INI parameter advtimeout to a value (in ms) and set this command to 0.

For example, the default full advertising frequency is approximately once per 2 seconds. It is recommended to set advtimeout = 20000, which will result in an advertising every 20 seconds which will result in a 90% power savings in the cost of advertising.

⚠ It is dangerous to turn off advertising in the network. When advertising is off, new motes can not join and existing motes can not rejoin the network after a reset. Turning off advertising is primarily used to save power, or may be useful in for specific use cases where it is desirable to prevent motes from joining the network. In most cases, it is best to allow advertising to remain under the control of the manager.

Request

Parameter	Type	Enum	Description
activate	INT8U	Advertisement State	Advertisement state

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
callbackId	INT32U		Callback ID

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_IN_PROGRESS	

	A command is still pending. Wait until a commandFinished notification is received for the previous command before retrying.

4.31 setCLIUser (0x3A)

Description

The *setCLIUser* command sets the password that must be used to log into the command line for a particular user role. The user roles are:

- Viewer - read-only access to non-sensitive information
- User - read-write access

This change is persistent.

Request

Parameter	Type	Enum	Description
role	INT8U	CLI User Roles	User role (see above)
password	INT8U[16]		Password

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_WRITE_FAIL	Flash write error, can't save new settings

4.32 setCommonJoinKey (0x42)

Description

The *setCommonJoinKey* command will set a new value for the common join key. The common join key is used to decrypt join messages only if the ACL is empty.

Request

Parameter	Type	Enum	Description
key	SEC_KEY		Common join key

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed

4.33 setDownstreamFrameMode (0x33)

Description

The *setDownstreamFrameMode* command tells the manager to shorten or extend the downstream slotframe. The base slotframe length will be multiplied by the *downFrameMultVal* for "normal" speed. For "fast" speed the downstream slotframe is the base length. Once this command is executed, the manager switches to manual mode and no longer changes slotframe size automatically. The response is a *callbackId*. A commandFinished notification with the *callbackId* is generated when the command propagation is complete.

Request

Parameter	Type	Enum	Description
frameMode	INT8U	Downstream Frame Mode	Downstream slotframe mode

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
callbackId	INT32U		Callback ID

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_IN_PROGRESS	A command is still pending. Wait until a commandFinished notification is received for the previous command before retrying.
RC_INVALID_ARGUMENT	The downFrameMultVal (as set by setNetworkConfig) is equal to 1, so changing the downstream frame mode would have no effect.

4.34 setIPConfig (0x44)

Description

The *setIPConfig* command sets the IPv6 prefix of the mesh network. Only the upper 8 bytes of the IPv6 address are relevant: the lower 8 bytes of the IPv6 address are ignored, and lower 8 bytes of the mask field are reserved and should be set to 0. This change is persistent.

Request

Parameter	Type	Enum	Description
ipv6Address	IPV6_ADDR		IPv6 address
mask	INT8U[16]		Subnet mask

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_WRITE_FAIL	Flash write error, can't save new settings

4.35 setLicense (0x38)

Description

The *setLicense* command has been deprecated in Manager >= 1.3.0. There is no longer a need to use a license to enable > 32 mote networks.

The *setLicense* command validates and updates the software license key stored in flash. Features enabled or disabled by the license key change will take effect after the device is restarted. If the *license* parameter is set to all 0x0s, the manager restores the default license. This change is persistent.

Request

Parameter	Type	Enum	Description
license	INT8U[13]		Software license key

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_VALIDATION_ERROR	The license key is not valid
RC_WRITE_FAIL	Flash write error, cannot save new settings

4.36 setNetworkConfig (0x1A)

Description

The *setNetworkConfig* command changes network configuration parameters. The response code indicates whether the changes were successfully applied. This change is persistent.

Generally, changes to network configuration will take effect when the manager reboots. Exceptions are detailed below:

- **Max Motes**: The new *maxMotes* value is used as soon as new motes try to join the network, but motes are not removed from the network if the value is set to a number lower than *numMotes*.

Request

Parameter	Type	Enum	Description
networkId	INT16U		Network ID
apTxPower	INT8S	Transmit Power	Access Point transmit power
frameProfile	INT8U	Frame Profile	The frame profile describes the length of the frames during network building and normal operation. All of the legacy frame profiles are mapped to the standard IP manager profile: Profile 1 (fast build, medium speed operation).
maxMotes	INT16U		The maximum number of motes allowed in the network (includes AP). The value can be 1-33 or 1-101, depending on the manager part number and the installed license (Mgr version < 1.3.0).
baseBandwidth	INT16U		Base bandwidth is the default bandwidth allocated to each mote that joins, defined as expected interval between packets, in ms. 0=no allocation.
downFrameMultVal	INT8U		Downstream frame multiplier is a multiplier for the length of the primary downstream frame. Valid values are 1, 2, or 4.
numParents	INT8U		Number of parents to assign each mote (1-4).
ccaMode	INT8U	CCA Mode	Indicates the mode for Clear Channel Assessment (CCA) in the network.
channelList	INT16U		Bitmap of channels in the whitelist to use for communication, all others blacklisted. Bit 0×0001 corresponds to channel 0 and bit 0×8000 corresponds to channel 15. (0=not used, 1=used). See the SmartMesh IP User's Guide section "Channel Blacklisting" for restrictions on the number of channels.
autoStartNetwork	BOOL		Deprecated - do not use
locMode	INT8U		Reserved

bbMode	INT8U	Backbone Frame Mode	Backbone frame mode (0=off,1=up,2=bidirectional)
bbSize	INT8U		Backbone frame size, in time slots (if bbmode=1, bbsize=1,2,4,8. If bbmode=2, bbsize=2)
isRadioTest	INT8U		Controls whether the manager boots up in radiotest mode.
bwMult	INT16U		Bandwidth over-provisioning multiplier: over-provision by value/100 (100-1000)
oneChannel	INT8U		Channel number for One Channel mode. (0-15; 255=OFF). This mode is used for rf testing only.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_INVALID_ARGUMENT	Validation of the network parameters failed
RC_WRITE_FAIL	Flash write error, cannot save new settings

4.37 setTime (0x36)

Description

This command has been deprecated, and should not be used in new designs. When the Manager restarts, it will start counting from 20:00:00 UTC July 2, 2002.

The *setTime* command sets the UTC time on the manager. This command may only be executed when the network is not running. If the *trigger* flag is false, the manager sets the specified time as soon as it receives the *setTime* command. When the manager receives a Time Pin trigger, it temporarily stores the current time. If a *setTime* request is received within a short period of time following the trigger, the manager calculates the delay since the trigger and adjust the time such that the trigger was received at the specified time value.

Request

Parameter	Type	Enum	Description
trigger	INT8U		0=set time immediately; 1=set time as using last timepin trigger
utcTime	UTC_TIME_L		Time to set on the Manager (as UTC microseconds)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed. The manager is ready to set the time.
RC_INVALID_ARGUMENT	One of the parameters was invalid
RC_VALIDATION_ERROR	Network is running, *setTime* command is disabled.

4.38 startNetwork (0x2D)

Description

The *startNetwork* command tells the manager to allow the network to start forming (begin accepting join requests from devices). The external application must issue the *startNetwork* command if the *autoStartNetwork* flag is not set (see *setNetworkConfig*).

This command has been deprecated and should not be used in new designs.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed
RC_IN_PROGRESS	The network is already started

4.39 subscribe (0x16)

Description

The *subscribe* command indicates that the manager should send the external application the specified notifications. It contains two filter fields:

- *filter* is a bitmask of flags indicating the types of notifications that the client wants to receive
- *unackFilter* allows the client to select which of the notifications selected in *filter* should be sent acknowledged. If a notification is sent as 'acknowledged', the subsequent notification packets will be queued while waiting for response.

Each subscription request overwrites the previous one. If an application is subscribed to data and then decides he also wants events he should send a *subscribe* command with both the data and event flags set. To clear all subscriptions, the client should send a subscribe command with the filter set to zero. When a session is initiated between the manager and a client, the subscription filter is initialized to zero.

The *subscribe* bitmap uses the values of the notification type enumeration. Some values are unused to provide backwards compatibility with earlier APIs.

Request

Parameter	Type	Enum	Description
filter	INT32U	Subscription filter	Subscription bitmap
unackFilter	INT32U	Subscription filter	Bitmap of notifications that should be sent using unacknowledged communications; 0=acknowledged, 1=unacknowledged. Unless the corresponding bit is set in the *filter* bitmap, the bit in the *unackFilter* has no meaning.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command successfully completed

RC_INVALID_ARGUMENT	Invalid subscription filter value

5 Notifications (0x14)

All notification packets have the packet type *Notification* (the packet type is part of the Serial API Header). The payload of the Serial API packet contains a notification structure. Each notification structure starts with a type field (see the Notification Types enumeration) indicating the type of notification and how the remainder of the payload can be deserialized.

Serial API Header	Notification type, 1 byte	Notification structure, N bytes, (see specific notification description)

If a notification is marked as 'acknowledged' in the **Control** field, it must be acknowledged by the client. This property can be controlled via the *subscribe* command. If a client receives an 'acknowledged' notification type that it does not understand, it must acknowledge the notification and ignore it. A notification acknowledgment has a **Control** field indicating an acknowledged response, packet type *Notification*, the same sequence number as a received notification and a single byte of payload with the response code *OK.*

Serial API Header	Response code OK, 1 byte = `0x00`

- Data Notifications contain upstream data sent by motes.
- IP Data Notifications contain upstream 6LoWPAN data sent by motes.
- Event Notifications contain network or system notifications sent by the Manager.
- Health Report Notifications contain health reports from the motes.
- Log Notifications contain diagnostic logs

5.1 blink Notification

The Blink Notification that will be generated by the manager is a regular data notification. In the blink notification, the user payload is encapsulated into a list of command id, length, value structures described below.

Blink notifications will only be made for devices listed in the Manager ACL.

Field	Type	Value	Description
command id	INT8U	0x94	Blink payload command id
length	INT8U		Length of the user payload
data	INT8U[]		User payload passed to the blink command

If the Blink command was called with the discovered neighbors flag set to 1, then the following structure will also be in the blink payload.

Field	Type	Value	Description
command id	INT8U	0x95	Reduced discovered neighbor command id
length	INT8U		Length of the discovered neighbors data
numNeighbors	INT8U		Number of discovered neighbors
neighbors	dscv_neighbor[]		List of reduced discovered neighbor structures, see below

Each neighbor is described by the *dscv_neighbor* structure:

Field	Type	Value	Description
moteId	INT16U		Mote ID of the heard neighbor
rssi	INT8S		RSSI of the heard neighbor

5.2 data Notification

Description

The *data* notification contains a header and a variable length array of binary data. The length of the data is determined based on the length of the notification.
The manager forwards all packets received on its IP address and non-manager ports as *data* notifications.

Notification Structure

Field	Type	Enum	Description
notifType	INT8U	Notification Type	Data payload (*data*)
timestamp	UTC_TIME_L	none	Time that the packet was generated at the mote
macAddress	MAC_ADDR	none	MAC address of the generating mote
srcPort	INT16U	none	Source port
dstPort	INT16U	none	Destination port
data	INT8U[]	none	Data payload

5.3 event Notifications

Description

Events describe system-level changes and various changes in the network topology. The *event* notification consists of:

- A unique event identifier,
- An event type, and
- An event detail structure that is specific to the event type.

Notification Structure

Field	Type	Enum	Description
notifType	INT8U	Notification Type	Notification type (*event*)
eventId	INT32U	none	Event ID
eventType	INT8U	Event Type	Event type
eventData	----	none	Event detail structure that is specific to the event type

5.3.1 commandFinished Event

Description

The *commandFinished* notification is sent when a command associated with the provided callback id finishes executing.

Notification Structure

Field	Type	Enum	Description
callbackId	INT32U	none	Callback ID that was returned in the response packet of the corresponding command
rc	INT8U	Command Finished Result	Command finished result code

5.3.2 pathCreate Event

Description

This notification is sent when the manager creates a connection (path) between two motes.

Notification Structure

Field	Type	Enum	Description
source	MAC_ADDR	none	MAC address of the source mote
dest	MAC_ADDR	none	MAC address of the destination mote
direction	INT8U	Path Direction	Path direction

5.3.3 pathDelete Event

Description

This notification is sent when the manager removes a connection (path) between two motes.

Notification Structure

Field	Type	Enum	Description
source	MAC_ADDR	none	MAC address of source mote
dest	MAC_ADDR	none	MAC address of destination mote
direction	INT8U	Path Direction	Path direction

5.3.4 ping Event

Description

This notification is sent when a reply is received from a mote ping.

Notification Structure

Field	Type	Enum	Description
callbackId	INT32U	none	The callback ID that was returned in the response packet associated with the ping mote request.
macAddress	MAC_ADDR	none	MAC address of mote pinged
delay	INT32U	none	Round trip delay in milliseconds or -1: ping timeout
voltage	INT16U	none	Voltage reported by mote (millivolts)
temperature	INT8S	none	Temperature reported by mote, in Celsius

5.3.5 networkTime Event

Description

The *time* notification is triggered by the client asserting the TIME pin or by calling the *getTime* command. This notification contains the time when the TIME pin was asserted (or the *getTime* command was processed) expressed as:

- *ASN*—The absolute slot number (the number of timeslots since " 7/2/2002 8:00:00 PM PST" if UTC is set on manager, otherwise since Jan 1, 1970)
- *Uptime*—The number of seconds since the device was booted
- *Unix time*—The number of seconds and microseconds since Jan 1, 1970 in UTC

Notification Structure

Field	Type	Enum	Description
uptime	INT32U	none	Time (sec) that the packet was generated (as uptime)
utcTime	UTC_TIME_L	none	Time that the packet was generated (as UTC)
asn	ASN	none	Absolute slot number
asnOffset	INT16U	none	ASN offset (in microseconds).

5.3.6 networkReset Event

Description

This notification is sent when the manager starts the network. This event has no *eventData* fields.

Notification Structure

Field	Type	Enum	Description

5.3.7 moteJoin Event

Description

This notification is sent when a mote joins the network.

Notification Structure

Field	Type	Enum	Description
macAddress	MAC_ADDR	none	Mote MAC address

5.3.8 moteCreate Event

Description

This event is sent when a mote joins the manager for the first time.

Notification Structure

Field	Type	Enum	Description
macAddress	MAC_ADDR	none	Mote MAC address
moteId	INT16U	none	Mote ID

5.3.9 moteDelete Event

Description

This notification is sent when a mote is deleted as a result of *moteDelete* command.

Notification Structure

Field	Type	Enum	Description
macAddress	MAC_ADDR	none	Mote MAC address
moteId	INT16U	none	Mote ID

5.3.10 moteLost Event

Description

This notification is sent when a mote's state changes to Lost, which indicates that the mote is not responding to downstream messages.

Notification Structure

Field	**Type**	**Enum**	**Description**
macAddress	MAC_ADDR	**none**	**Mote MAC address**

⚠ The *moteLost* event is not generated when a mote rejoins the network, however if the *motest* trace is on, the manager will mark a transition between **Lost** and **Negotiating1** when it receives the mote's join request.

5.3.11 moteOperational Event

Description

This notification is sent when a mote that joins the network becomes operational.

Notification Structure

Field	Type	Enum	Description
macAddress	MAC_ADDR	none	Mote MAC address

5.3.12 moteReset Event

Description

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

Notification Structure

Field	Type	Enum	Description
macAddress	MAC_ADDR	none	Mote MAC address

5.3.13 packetSent Event

Description

The *packetSent* notification is generated when client's packet is removed from manager's queue and sent into the wireless network.

Notification Structure

Field	Type	Enum	Description
callbackId	INT32U	none	Callback ID that was returned in the response packet of the corresponding command
rc	INT8U	none	Result code

5.3.14 invalidMIC Event

Description

The invalidMIC event is generated when a packet that the manager receives from a mote in the network fails decryption. This notification is available in Manager >= 1.4.1.

Notification Structure

Field	Type	Enum	Description
macAddress	MAC_ADDR	none	Mote MAC address

5.3.15 joinFailed Event

Description

The joinFailed event is generated when a mote sends a join request to the manager but the request can not be validated. This notification is available in Manager >= 1.4.1.

Notification Structure

Field	Type	Enum	Description
macAddress	MAC_ADDR	none	Mote MAC address
reason	INT8U	Join Failure Reasons	Reason for the join failure

5.4 healthReport Notification

Description

The *healthReport* notifications include the raw payload of health reports received from devices. The payload contains one or more specific health report messages. Each message contains an identifier, length and variable-sized data. The individual *healthReport* message structures are defined below.

Notification Structure

Field	Type	Enum	Description
notifType	INT8U	Notification Type	Notification type (*healthReport*)
macAddress	MAC_ADDR	none	The MAC address of the mote from which the health report was received
payload	INT8U[]	none	Variable length payload of one or more health report messages

5.4.1 Health Report Messages

Device Health Report

The Device Health Report reports on the device's statistics accumulated since the last device health report.

Field	Type	Value	Description
id	INT8U	`0x80`	Device Health Report identifier
length	INT8U	`0x18`	Length of the remainder of the Device Health Report
charge	INT32U		Lifetime charge consumption (in mC)
queueOcc	INT8U		Mean and max queue occupancy. Bits 0-3 are the mean queue occupancy, and bits 4-7 are the max queue occupancy.
temperature	INT8S		Mote temperature (in degrees C)
batteryVoltage	INT16U		Mote battery voltage (in mV)
numTxOk	INT16U		Number of packets sent from NET to MAC
numTxFail	INT16U		Number of packets not sent due to congestion or failure to allocate a packet
numRxOk	INT16U		Number of received packets

numRxLost	INT16U		Number of packets lost (discarded by NET layer due to misc errors)
numMacDropped	INT8U		Number of packets dropped by MAC (due to retry count or age or no route)
numTxBad	INT8U		Transmit failure counter for bad link
badLinkFrameId	INT8U		Frame id of link with the worst performance over the last health report interval
badLinkSlot	INT32U		Slot of link with the worst performance over the last health report interval
badLinkOffset	INT8U		Offset of link with the worst performance over the last health report interval
numNetMicErr	INT8U		Number of incoming packets terminated by this mote that fail decryption Note: With mote version 1.4.1 or later
numMacMicErr	INT8U		Number of incoming packets that fail authentication Note: With mote version 1.4.1 or later
numMacCrcErr	INT8U		number of incoming packets with MAC-layer CRC errors. The intent of this field is to indicate the presence of unusual traffic or jamming that is interfering with the network Note: With mote version 1.4.1 or later

Neighbors' Health Report

The Neighbors' Health Report contains a report of current statistics about communication with each neighbor of a mote.

Field	Type	Value	Description
id	INT8U	`0x81`	Neighbor Health Report identifier
length	INT8U		Length of the remainder of the message
numItems	INT8U		Number of *neighborHRData* structures in this message
neighbors	neighborHRData[]		Sequence of *numItems neighborHRData* structures

neighborHRData structure

Field	Type	Value	Description
neighborId	INT16U		Neighbor Mote ID
neighborFlag	INT8U		See Neighbor Flags
rssi	INT8S		RSSI of neighbor
numTxPackets	INT16U		Number of transmitted packets

numTxFailures	INT16U		Number of failed transmission
numRxPackets	INT16U		Number of received packets

Discovered Neighbors Health Report

The Discovered Neighbor Health Report contains a list of neighbors discovered in this health report interval.

Field	Type	Value	Description
id	INT8U	`0x82`	Discovered Neighbors Health Report identifier
length	INT8U		Length of the remainder of the message
numJoinParents	INT8U		Number of parent motes
numItems	INT8U		Number of discovered neighbor structures in this message
discoveredNeighbors	discoveredNeighborData[]		Sequence of *numItems discoveredNeighborData* structures

discoveredNeighborData structure

Field	Type	Value	Description
neighborId	INT16U		Neighbor Mote ID
rssi	INT8S		RSSI of neighbor
numRx	INT8U		Number of times a neighbor was heard

Extended Health Reports

Some Health Reports are categorized as Extended Health Report notifications. These Health Reports are not processed by the manager and specific payloads are forwarded as sub-types of the Extended Health Report notification. The generic format of Extended Health Reports is:

Field	Type	Value	Description
id	INT8U	`0x91`	Extended Health Report identifier
length	INT8U		Length of the remainder of the message
extType	INT8U		Specific type of the Extended Health Report
extLength	INT8U		Length of the specific Extended Health Report payload
extPayload	INT8U[]		Payload of the health report

RSSI Report

The RSSI Report contains a list of channels with RSSI and transmit statistics for each. The RSSI Report can be generated by IP Stack versions 1.4 or later.

Field	Type	Value	Description
id	INT8U	`0x91`	Extended Health Report identifier
length	INT8U		Length of the remainder of the message
extType	INT8U	`1`	Extended Health Report type, RSSI Report
extLength	INT8U	`75`	Length of the RSSI Report payload
rssiReport	rssiReport[15]		Payload of the RSSI Report

rssiReport structure

Field	Type	Value	Description
idleRssi	INT8U		Average RSSI measured during idle listens
txUnicastAttempts	INT16U		Number of unicast attempts on the channel
txUnicastFailures	INT16U		Number of missed ACKs on the channel

5.5 ipData Notification

Description

The *ipData* notification contains full IP packet sent by the mote, including 6LoWPAN header, UDP header, and the UDP payload. Manager generates this notification when it receives packet from a mote with destination other than manager's own IP address. The size of the *data* field can be calculated by subtracting the fixed header size (up to *macAddress)* from the size of overall notification packet.

Notification Structure

Field	Type	Enum	Description
notifType	INT8U	Notification Types	Notification type (*ipData*)
utcTime	UTC_TIME_L	none	UTC timestamp of time that the packet was generated (at the mote)
macAddress	MAC_ADDR	none	MAC address of the generating mote
data	INT8U[]	none	6LoWPAN packet

6LoWPAN Packet Format

Section	Description	Length (Bytes)	Encoding
6LoWPAN IP Header	LOWPAN_IPHC	2	011.11.1.10:C.S.ss.M.D.dd • 011 - indicates LOWPAN header • 11 - Traffic Class and Flow Label are elided • 1 - Next Header field is compressed • 10 - Hop Limit is compressed (limit 64) • C context identifier (0 = elided for current version of LBR) • S/D source/destination compressed (1) • s/d source/destination mode: 00 – full (16B - only used for external destination upstream), 11 – elided / compressed (only for destination if M=1) (0/1B) • M multicast (0 - multicast addresses are note used for current version of LBR)
		0/1	SSSS.DDDD

	Context ID Extension		• S, D – source / destination
	Source Address	0/16	always elided for packets from the mesh
	Destination Address	0/1/16	always elided for devices in the mesh, 16 bytes if destination is external to the mesh
UDP	Header	0/1	111101.S.D • S/D– source/dest. port is compressed (1)
	Ports	1/3/4	If SD in UDP header is 11, both ports are in range F0Bx and ports are encoded as 1 byte, ssss.dddd • s, d – source / destination If SD = 10, source port is in range F0xx, and encoded as 1 byte. 2-byte destination port is carried inline If SD = 01, destination port is in range F0xx and encoded as 1 byte. 2-byte source port is carried inline If SD = 00, each 2-byte port is carried inline
Payload	User payload	Variable	

5.6 log Notification

Description

A *log* notifications is generated in response to the *getLog* command. Each *log* notification contains a message from the mote's log.

Notification Structure

Field	Type	Enum	Description
notifType	INT8U	Notification Types	Notification type (*log*)
macAddress	MAC_ADDR	none	MAC address of notification source
logMsg	INT8U[]	none	Log message

6 Definitions

This section lists constants and pre-defined values used in the API structures.

6.1 Packet and Command Types

Each message is assigned a packet type. The first several packet types are used for the Serial API Protocol handshake. Unspecified packet types are reserved for compatibility or future use.

Name	Value	Description
(Null)	0x00	Null packet (reserved)
Hello	0x01	Client Hello message
HelloResponse	0x02	Manager response to client Hello
MgrHello	0x03	Manager Hello message
Notification	0x14	Notification packet
reset	0x15	Reset the system, network, or a mote
subscribe	0x16	Subscribe to notifications
getTime	0x17	Return the current manager UTC time and absolute slot number (ASN)
setNetworkConfig	0x1A	Set network configuration parameters
clearStatistics	0x1F	Clear accumulated network statistics
exchangeMoteJoinKey	0x21	Send a new join key to a mote
exchangeNetworkId	0x22	Send a new network ID to a mote
radiotestTx	0x23	Command for testing the radio
radiotestRx	0x25	Command for testing the radio
getRadiotestStatistics	0x26	Command for testing the radio
setACLEntry	0x27	Add a new ACL entry or update an existing entry
getNextACLEntry	0x28	Iterate through the ACL entries
deleteACLEntry	0x29	Remove a mote from the ACL
pingMote	0x2A	Send a ping (echo request) to a mote

getLog	0x2B	Retrieve diagnostic logs from a mote
sendData	0x2C	Send a data packet to a mote
startNetwork	0x2D	Start network formation (deprecated - do not use)
getSystemInfo	0x2E	Return system-level information about the hardware and software
getMoteConfig	0x2F	Retrieve mote configuration parameters
getPathInfo	0x30	Get information about communication between two motes
getNextPathInfo	0x31	Iterate through a mote's neighbors
setAdvertising	0x32	Turn on or off advertising
setDownstreamFrameMode	0x33	Shorten or extend the downstream slotframe
reserved	0x34	reserved
getManagerStatistics	0x35	Get manager API statistics
setTime	0x36	Set the UTC time on the manager
getLicense	0x37	Return the current license key (deprecated in Mgr >= 1.3.0)
setlLicense	0x38	Update the software license key (deprecated in Mgr >= 1.3.0)
setCLIUser	0x3A	Update CLI logins
sendIP	0x3B	Send IP data to a mote (via 6LowPAN)
reserved	0x3C	Reserved
restorefFactoryDefaults	0x3D	Restore default configuration and clear the ACL
getMoteInfo	0x3E	Get mote statistics
getNetworkConfig	0x3F	Retrieve network configuration parameters
getNetworkInfo	0x40	Get network statistics
getMoteConfigById	0x41	Retrieve a mote's configuration by mote ID
setCommonJoinKey	0x42	Set new value for common join key
getIPConfig	0x43	Return manager's IP configuration parameters
setIPConfig	0x44	Set manager's IP configuration parameters
deleteMote	0x45	Delete a mote from the Manager's database
getMoteLinks	0x46	Get information about mote's links

6.2 Notification Types

Name	Value	Description
event	1	Event notification
log	2	Log notification
data	4	Data payload notification
ipData	5	6lowpan packet notification
healthReport	6	Health report notification

⚠ In the Manager API, there is a single notification command type (0x14), and the first field in the notification tells you what kind of notification it is. In the Mote API, each notification has its own command type. The first field in the event notification tells you what kind of event it is.

6.3 Subscription Filters

Filter	Value
Event	0x02
Log	0x04
Data	0x10
IP Data	0x20
Health Reports	0x40

6.4 Event Types

Name	Value	Description
moteReset	0	A mote reset
networkReset	1	The network was reset
commandFinished	2	A command has completed execution
moteJoin	3	A mote joined the network
moteOperational	4	A new mote was configured and is now operational

moteLost	5	A mote is no longer communicating in the network
networkTime	6	Contains the network uptime (in response to a getTime command)
pingResponse	7	A reply was received from a mote ping
reserved	9	reserved
pathCreate	10	A path was created
pathDelete	11	A path was deleted
packetSent	12	A packet was sent
moteCreate	13	A mote was created
moteDelete	14	A mote was deleted
joinFailed	15	A join request could not be processed
invalidMIC	16	A packet from a mote failed decryption

6.5 Response Codes

Name	Value	Description
RC_OK	0	The application layer has processed the command correctly
RC_INVALID_COMMAND	1	Invalid command
RC_INVALID_ARGUMENT	2	Invalid argument
RC_END_OF_LIST	11	End of list is returned when an iteration reaches the end of the list of objects
RC_NO_RESOURCES	12	Reached maximum number of items
RC_IN_PROGRESS	13	Operation is in progress
RC_NACK	14	Negative acknowledgment
RC_WRITE_FAIL	15	Flash write failed
RC_VALIDATION_ERROR	16	Parameter validation error
RC_INV_STATE	17	Object has inappropriate state
RC_NOT_FOUND	18	Object is not found
RC_UNSUPPORTED	19	The operation is not supported

6.6 Frame Profile

Name	Value	Description
Profile_01	1	Fast network build, medium speed network operation

6.7 Advertisement State

Name	Value	Description
on	0	Advertisement is on
off	1	Advertisement is off or slow*

* Motes 1.4.1 or later allow for slow advertisements. Adv rate set via the *advtimeout* INI parameter.

6.8 Downstream Frame Mode

Name	Value	Description
normal	0	Normal downstream bandwidth
fast	1	Fast downstream bandwidth

6.9 Network State

Name	Value	Description
operational	0	Network is operating normally
radiotest	1	Manager is in radiotest mode
notStarted	2	Waiting for startNetwork API command
errorStartup	3	Unexpected error occurred at startup
errorConfig	4	Invalid or not licensed configuration found at startup
errorLicense	5	Invalid license file found at startup

6.10 Mote State

Name	Value	Description
lost	0	Mote is not currently part of the network
negotiating	1	Mote is in the process of joining the network
operational	4	Mote is operational

6.11 Reset Type

Name	Value	Description
resetSystem	0	Reset the system
resetMote	2	Reset the mote

6.12 Backbone Frame Mode

Name	Value	Description
off	0	Backbone frame is off
upstream	1	Backbone frame is activated for upstream frames
bidirectional	2	Backbone frame is activated for both upstream and downstream frames

6.13 Path Filter

Name	Value	Description
all	0	All paths
upstream	1	Upstream paths

6.14 Path Direction

Name	Value	Description
none	0	No path
unused	1	Path is not used
upstream	2	Upstream path
downstream	3	Downstream path

6.15 Packet Priority

Name	Value	Description
Low	0	Default packet priority
Medium	1	Higher packet priority
High	2	Highest packet priority

6.16 Command Finished Result

Name	Value	Description
OK	0	Command completed successfully
nack	1	Command not acknowledged
commandTimeout	2	Command timed out

6.17 Transmit Power

Transmit Power is a signed byte (INT8S) with the values 0, 8.

6.18 CCA Mode

Name	Value	Description
off	0	CCA disabled
energy	1	Energy detect
carrier	2	Carrier detect
both	3	Energy detect and Carrier detect

6.19 Link Flags

Flag	Value
Transmit	0x01
Receive	0x02
Shared	0x04
Reserved	0x08
Join	0x10
Advertisement	0x20
Discovery	0x40
No path failure detection	0x80

6.20 Neighbor Flags

Flag	Value
Existing path failure condition	0x01

6.21 CLI User Roles

Name	Value	Description
viewer	0	*Viewer*-role user has read-only access to non-sensitive network information
user	1	*User*-role user has read-write privileges

6.22 Radio Test Types

Name	Value	Description
packet	0	Transmit packets
cm	1	Continuous modulation
cw	2	Continuous wave

pkcca	3	Packet test with clear channel assessment (CCA) enabled

6.23 Join Failure Reasons

Name	Value	Description
counter	0	The join packet reused an already used join counter
notOnACL	1	The mote is not listed on the ACL
authentication	2	The join request could not be decrypted. Generally, this means the request was encrypted with a join key that did not match the key in the ACL.
unexpected	3	An unexpected error occurred while processing the join request