SmartMesh IP Mote Serial 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 revision
2	08/10/2012	Updated radiotest API
3	03/18/2013	Numerous small changes
4	10/22/2013	Document retitled
5	04/04/2014	Updated and clarified radiotest commands;
6	10/28/2014	Included command IDs in titles; Added Station ID to extended testRadio commands; Clarified Notification structure
7	04/22/2015	Added advKey to the setParameter commands; Other minor changes
8	12/03/2015	Added settings for EN 300 328 compliance; Added appVer parameter; Other minor changes
9	11/07/2016	Clarified description for getParameter<txPower> and setParameter<txPower>; Added blink, getParameter<entropy> and stopSearch commands

2 Introduction

This guide describes the commands used by an external processor to communicate with the SmartMesh IP mote through its API serial port. The API is intended for machine-to-machine communications (*e.g.* a sensor application talking to the mote).

In contrast, the command line interface (CLI) is intended for human interaction with a mote, *e.g.* during development, or for interactive troubleshooting. See the SmartMesh IP Mote CLI Guide for details on that interface.

3 Mote Serial Modes

The CLI UART supports a single operating mode:

- Mode 0: 9600 baud, not HDLC encoded,
  - 2-wire interface: only UART_RX and UART_TX signals are used

The API UART supports 2 operating modes, as configured in the device *fuse table*:

- Mode 2: 9600 or 115.2K baud, HDLC encoded
  - 6-wire interface: All UART signals are used
- Mode 4: 9600 or 115.2K baud, HDLC encoded
  - 4-wire interface: TX, RX, UART_TX_CTSn, UART_TX_RTSn signals are used.

Please see the relevant datasheet for your device for details on signal timing.

The fuse table for the LTC5800-IPM is normally developed as part of the board level design process. Either mode 2 or mode 4 may be used on the API port, at either baud rate. For modularly certified products such as the LTP5901-IPM, or the Starter Kit mote (DC9003A-B) the fuse table has been pre-programmed for mode 4 at 115.2 Kbps and cannot be changed.

4 Protocol

The Mote Serial API provides OEMs with a well-defined, easy-to-integrate Application Programming Interface (API) via a mote serial interface. Through this interface, OEMs have access to the rich capabilities of the mote and the network. The API includes commands for configuring and controlling the mote, querying mote settings, sending and receiving data over the wireless mesh network, and testing RF functions.

The connection between the mote and microprocessor is a wired serial connection. Refer to the mote product datasheet for details and specifications on voltage signaling levels, serial handshake definition, and signal timing. The Mote Serial API described in this document operates over this interface.

4.1 Data Representation

4.1.1 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

4.1.2 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

4.1.3 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).

4.2 Packet Format

4.2.1 HDLC Packet Encapsulation

HDLC protocol is used for all API communication between mote and serial microprocessor. All packets are encapsulated in HDLC framing described in RFC1662. Note that the interface does not comply with all aspects of RFC1662 - only framing, i.e. start and stop flags, escaping of flags in payload, and FCS are used from RFC1662. Packets start and end with a 0x7E flag, and contain a 16-bit CRC-CCITT Frame Check Sequence (FCS). Note that packets do not contain HDLC Control and Address fields that are mentioned in RFC1662.

Start Flag (Byte 0)	HDLC Payload (Bytes 1-n)	FCS (Bytes n+1, n+2)	End Flag (Byte n+3)
0x7E	HDLC escaped API payload	(2 bytes)	0x7E

⚠ Some products may require an extra 0x7E start delimiter for proper operation at high bit rates. Refer to specific product datasheets for details.

Byte-stuffing is used to escape the Flag Sequence (0x7E) and Control Escape (0x7D) bytes that may be contained in the Payload or FCS fields. Async-Control-Character-Map (ACCM) mechanism is not used, so all other bytes values can be sent without an escape. After FCS computation, the transmitter examines the entire frame between the starting and ending Flag Sequences. Each 0x7E and 0x7D (excluding the start and end flags) is then replaced by a two-byte sequence consisting of the Control Escape (0x7D) followed by the XOR result of the original byte and 0x20, i.e.:

- 0x7D -> 0x7D 0x5D
- 0x7E -> 0x7D 0x5E

HDLC Encoding Example

Assume the following payload needs to be sent (command, length, flags, data):

Payload
03 | 07 | 02 | 00 00 00 00 03 00 7D

Calculate and append FCS:

Payload	FCS
03 07 02 00 00 00 00 03 00 **7D**	9A B2

Perform byte stuffing. In this case, one occurrence of 0x7D needs to be escaped:

Payload and FCS (stuffed)
03 07 02 00 00 00 00 03 00 **7D 5D** 9A B2

Finally add start and end flags. The packet is ready for transmission:

Flag	Payload and FCS (stuffed)	Flag
7E	03 07 02 00 00 00 00 03 00 **7D 5D** 9A B2	7E

HDLC Decoding Example

Assume that the following packet was received:

Flag	Payload and FCS (stuffed)	Flag
7E	04 03 01 00 03 00 **7D 5E** A2 91	7E

Remove start and end flags:

Payload and FCS (stuffed)
04 03 01 00 03 00 **7D 5E** A2 91

Perform byte un-stuffing. In this case, one sequence of 0x7D 0x5E is replaced by 0x7E. The last two bytes can now be treated as FCS:

Payload	FCS
04 03 01 00 03 00 **7E**	A2 91

Finally, check that FCS of the payload matches the last two bytes received. If it does, the payload is valid and should be processed.

Payload
04 03 01 00 03 00 **7E**

4.2.2 HDLC Payload Contents

All packets sent on serial interface have a common API header followed by API payload. A flag in the header identifies payload as Request or Response.

Start	API Header	API Payload	FCS	End
0x7E	Command | Length | Flags	Response code (responses only) | Message Payload	(2 Bytes)	0x7E

HDLC Payload Format

API Header

The API header contains the following fields:

Field	Type	
Command Id	INT8U	Command identifier (see Commands)
Len	INT8U	Length of API Payload (excludes this header)
Flags	INT8U	Packet Flags

Flags field

Flags is an INT8U field containing the following fields:

Bit	Description
0 (LSB)	0=Request, 1=Response
1	Packet id
2	*Reserved, set to 0*
3	Sync
4	*Reserved, set to 0*
5	*Reserved, set to 0*
6	*Reserved, set to 0*
7 (MSB)	*Reserved, set to 0*

Request/Response flag

The Request/Response flag identifies a packet as containing request or response payload. The payload of each command and notification is unique and is defined in the sections below.

Packet id

Packet id is a 1-bit sequence number that is used to ensure reliable, in-order processing of packets.

The sender must toggle the **Packet id** field if a new packet is sent, and leave the field unchanged for retransmissions. The receiver should track the received **Packet id**. If a new packet is received, it ts processed and the response contains a copy of **Packet id**. If a duplicate **Packet id** is received, a cached copy of the response is sent.

The mote treats a repeat of a packet ID as a duplicate packet. It will drop the incoming command (even if the command itself differs from the previous command with the same packet ID) and respond with a cached response to the previous command.

Sync flag

The Sync bit is used to reset sequence numbers on the serial link.

The first request packet from the mote will have the Sync flag set. This lets the serial microprocessor know that the device booted up and is establishing communication. Similarly, the first request from the serial microprocessor should contain the Sync flag.

API Payload

The API Payload portion of the packet contains request or response payloads listed in the Commands and Notifications sections.

For responses, the first payload byte will always be a response code, which is not included in the length count.

Maximum packet size

Both the request and response packets have a maximum HDLC payload size of 128 bytes. This does not include start/stop delimiters or frame checksum. Octet-stuffing escape sequences do not count against this limit.

4.3 Communication Between Mote and Microprocessor

4.3.1 Acknowledged Link

All packets sent across the serial link must be acknowledged. The acknowledgement must be received before another serial packet may be sent. This applies to packets initiated by either the mote or the microprocessor. To allow the receiving side to distinguish between new and retransmitted packets, the sender must toggle th e **Packet id** field if a new packet is sent or if the sender receives a response with RC_NO_RESOURCES code. Packet Id should stay unchanged if the sender does not get any response.

The mote follows the same rules for packet re-transmission.

4.3.2 Guidelines for Forward-Compatible Clients

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

- Payloads may be extended to include new fields; new fields will either be added at the end or in place of reserved bytes
- Existing fields may become deprecated (but will not be removed)
- New commands and notifications may be added
- New alarms and events may be added
- New response codes may be added

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

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

If the protocol changes in any other incompatible way, the protocol version reported via *getParameter<moteInfo>* will be changed.

5 Commands

The following sections describe API payload of commands supported by mote. Note that the API header bytes are not listed.

API Header	API Payload

Command contents

5.1 blink (0x2E)

Description

Send a packet into the network without joining. The mote searches for a network and sends the packet. Optionally, the list of neighbors discovered during the search process is also sent, up to a maximum of four neighbors. If the discovered neighbor list is not included, the payload maximum size is 73B, and if the discovered neighbors are included, the maximum size is 58B.

Upon receiving a *blink* command, the mote will transition to the *blink* state and start searching for advertisements. When it hears an advertisement, it synchronizes and continues listening briefly in efforts to discover more neighbors. After this short timeout, the mote immediately sends the data packet to one of the discovered neighbors. If the *blink* command is called repeatedly to send consecutive packets, the mote does not search again unless the discovered neighbor list is requested.

When the mote successfully sends the packet, a txDone notification will be sent with *status* set to 0. If the mote cannot send the packet, e.g. if 60 seconds elapse without receiving a MAC-layer acknowledgement, a txDone notification is sent with *status* set to 1.

For Blink packets, the mote can only accept a single packet at a time. To send multiple packets, the application must wait for the txDone notification. The mote will return to low-power sleep when 60 seconds elapse without any MAC-layer acknowledgements, so to prevent the mote from sleeping, the application should send the packets much faster than this 60 second timeout. See the SmartMesh Embedded IP Manager API Guide for details on the manager-side blink notification.

The mote will stay in the *blink* state unless it is reset, OR a *join* command is issued. This was done so that the *getTime* command can be called at any time after at least one blink packet has been sent. Note that the clock is free-running since its last packet sent and will drift, however can be used as a relatively accurate clock source for many applications, if sending packets once a day for example.

Request

Parameter	Type	Enum	Description
fIncludeDscvNbrs	INT8U	None	Whether to include the discovered neighbor list in Blink packet. Set to 1 to include discovered neighbors, 0 otherwise.
payload	INT8U[]	None	Payload of the packet

Response

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code

Response Codes

Code	Description
RC_OK	Packet accepted
RC_INVALID_STATE	Another Blink packet is in the queue to be sent
RC_INVALID_LENGTH	The payload is too long

5.2 bindSocket (0x17)

Description

Bind a previously opened socket to a port. When a socket is created, it is only given a protocol family, but not assigned a port. This association must be performed before the socket can accept connections from other hosts.

Request

Parameter	Type	Enum	Description
socketId	INT8U	none	Socket ID to bind
port	INT16U	none	Port to bind to the socket

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation completed successfully
RC_NOT_FOUND	Invalid socket ID

5.3 clearNV (0x10)

Description

The *clearNV* command resets the mote's non-volatile memory (NV) to its factory-default state. See User Guide for detailed information about the default values. Since many parameters are read by the mote only at power-up, this command should be followed up by mote reset.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Return code

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Flash operation failed

5.4 closeSocket (0x16)

Description

Close the previously open socket.

Request

Parameter	Type	Enum	Description
socketId	INT8U	none	Socket ID

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Socket successfully closed
RC_NOT_FOUND	Socket ID not found

5.5 disconnect (0x07)

Description

The *disconnect* command requests that the mote initiate disconnection from the network. After disconnection completes, the mote will generate a disconnected event, and proceed to reset. If the mote is not in the network, the disconnected event will be generated immediately. This command will return an error if the mote is not presently joining or joined to a network.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted

5.6 getParameter (0x02)

5.6.1 getParameter<appInfo>

Description

Get the application (as opposed to the network stack) version information.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (appInfo)

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response Code
paramId	INT8U	Parameters	Parameter ID (appInfo)
vendorId	INT16U	none	Vendor ID
appId	INT8U	none	Application ID
appVer	APP_VER	none	Application Version

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.2 getParameter<antGain>

Description

The *getParameter<antGain>* command retrieves the antenna gain used by the system (to properly calculate radiated power). Defaults to 2 dBi.

Note: This parameter is available in devices running mote software >=1.4.x

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*antGain*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code
paramId	INT8U	Parameters	Parameter ID (*antGain*)
antGain	INT8S	none	Antenna gain in dBi

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.3 getParameter<autoJoin>

Description

This command allows the microprocessor to retrieve the current *autoJoin* setting.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*autoJoin*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*autoJoin*)
autoJoin	BOOL		0=do not auto join; 1=auto join

Response Codes

Code	Description
RC_OK	The command completed successfully

5.6.4 getParameter<charge>

Description

The *getParameter<charge>* command retrieves the charge consumption of the mote since the last reset.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*charge*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code
paramId	INT8U	Parameters	Parameter ID (*charge*)
qTotal	INT32U	none	Charge since last reset (millicoulombs)
upTime	INT32U	none	Time since reset (seconds)
tempInt	INT8S	none	Temperature (integral part, in °C)
tempFrac	INT8U	none	Temperature (fractional part, in 1/255 of °C)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.5 getParameter<entropy>

Description

The *getParameter<entropy>* command may be used to retrieve a 16-byte block of random data. The data is obtained from thermal noise in the LTC5800 receive signal chain with the radio front-end disabled - as such, it can only be called when the mote is in the *idle* state. Thus while it is suitable for cryptographic operations, it is recommended to be used as a seed for a DRBG because of this limitation.

This parameter is available in devices running mote software 1.4 or later.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (entropy)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code
paramId	INT8U	Parameters	Parameter ID (entropy)
entropy	INT8U[16]	none	Entropy

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_INVALID_STATE	The mote is in invalid state to retrieve entropy

5.6.6 getParameter<euCompliantMode>

Description

The getParameter<euCompliantMode> command may be used to retrieve the EN 300 328 compliance mode that is used by the device. When enabled, the mote may skip some transmit opportunities to remain within average power limits. Motes below +10 dBm radiated power do not need to duty cycle to meet EN 300 328 requirements.

Note: This parameter is available in devices running mote software >=1.4.x

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*euCompliantMode*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code
paramId	INT8U	Parameters	Parameter ID (*euCompliantMode*)
compMode	INT8U	none	EN 300 328 compliance mode. 0=off, 1=on

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.7 getParameter<eventMask>

Description

getParameter<eventMask> allows the microprocessor to read the currently subscribed-to event types.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*eventMask*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*eventMask*)
eventMask	INT32U	Events	Event mask. 1=subscribed, 0=unsubscribed

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.8 getParameter<ipv6Address>

Description

This command allows the microprocessor to read IPV6 address assigned to the mote. Before the mote has an assigned address it will return all 0s.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*ipv6Address*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*ipv6Address*)
ipv6Address	IPV6_ADDR	none	IPV6 address

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.9 getParameter<joinDutyCycle>

Description

This command allows user to retrieve current value of *joinDutyCycle* parameter.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*joinDutyCycle*)

Response

Parameter	Type	Enum	Description
rc	INT8U		Response code
paramId	INT8U	Parameters	Parameter ID (*joinDutyCycle*)
joinDutyCycle	INT8U		Duty cycle (0-255), where 0=0.2% and 255=99.8%

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.10 getParameter<macAddress>

Description

This command returns the MAC address of the device. By default, the MAC address returned is the EUI64 address of the device assigned by mote manufacturer, but it may be overwritten using the *setParameter<macAddress>* command.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*macAddress*)

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response code
paramId	INT8U	Parameters	Parameter ID (*macAddress*)
macAddress	MAC_ADDR	none	MAC address of the device

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.11 getParameter<shortAddress>

Description

This command retrieves the mote's Mote ID. If the mote is not in the network, value of 0 is returned.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*moteId*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*moteId*)
moteId	INT16U		Mote ID

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.12 getParameter<moteInfo>

Description

The *getParameter<moteInfo>* command returns static information about the mote's hardware and network stack software.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*moteInfo*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*moteInfo*)
apiVersion	INT8U		Version of the API protocol
serialNumber	INT8U[8]		Serial number of the device
hwModel	INT8U		Hardware model
hwRev	INT8U		Hardware revision
swVerMajor	INT8U		Network stack software version: major
swVerMinor	INT8U		Network stack software version: minor
swVerPatch	INT8U		Network stack software version: patch
swVerBuild	INT16U		Network stack software version: build
bootSwVer	INT8U		Bootloader software version

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.13 getParameter<moteStatus>

Description

The *getParameter<moteStatus>* command is used to retrieve current mote state and other dynamic information.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*moteStatus*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*moteStatus*)
state	INT8U	Mote States	Mote state
reserved_1	INT8U[3]		This field should be ignored
numParents	INT8U		Number of parents
alarms	INT32U	Alarms	Current alarms (bitmap)
reserved_2	INT8U		This field should be ignored

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.14 getParameter<netInfo>

Description

The *getParameter<networkInfo>* command may be used to retrieve the mote's network-related parameters.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*netInfo*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*netInfo*)
macAddress	MAC_ADDR	none	MAC address of the device
moteId	INT16U	none	Mote ID
networkId	INT16U	none	Network ID
slotSize	INT16U	none	Slot size (microseconds)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.15 getParameter<networkId>

Description

This command returns the network id stored in mote's persistent storage.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*networkId*)

Response

Parameter	Type	Enum	Description
rc	INT8U		Response code
paramId	INT8U	Parameters	Parameter ID (*networkId*)
networkId	INT16U		Network Identifier

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.16 getParameter<OTAPLockout>

Description

This command reads the current state of OTAP lockout, i.e. whether over-the-air upgrades of software are permitted on this mote.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*OTAPLockout*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code
paramId	INT8U	Parameters	Parameter ID (*OTAPLockout*)
mode	BOOL		0=OTAP allowed, 1=OTAP not allowed

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.17 getParameter<pwrSrcInfo>

Description

This command allows the microprocessor to read a mote's power source settings.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*powerSrcInfo*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*powerSrcInfo*)
maxStCurrent	INT16U	none	Maximum steady-state current in µA; 0xFFFF=no limit
minLifetime	INT8U	none	Minimum lifetime, in months; 0=no limit
currentLimit_0	INT16U	none	Current limit, in µA; 0=limit entry empty
dischargePeriod_0	INT16U	none	Discharge period, in seconds
rechargePeriod_0	INT16U	none	Recharge period, in seconds
currentLimit_1	INT16U	none	Current limit, in µA; 0=limit entry empty
dischargePeriod_1	INT16U	none	Discharge period, in seconds
rechargePeriod_1	INT16U	none	Recharge period, in seconds
currentLimit_2	INT16U	none	Current limit, in µA; 0=limit entry empty
dischargePeriod_2	INT16U	none	Discharge period, in seconds
rechargePeriod_2	INT16U	none	Recharge period, in seconds

Response Codes

Code	Description

RC_OK	Command completed successfully

5.6.18 getParameter<routingMode>

Description

This command allows the microprocessor to retrieve the current routing mode of the mote.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*routingMode*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code
paramId	INT8U	Parameters	Parameter ID (*routingMode*)
routingMode	BOOL		Routing mode (0=enable routing, 1=disable routing)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.19 getParameter<testRadioRxStats>

Description

The *getParameter<testRadioRxStats>* command retrieves statistics for the latest radio test performed using the testRadioRx command. The statistics show the number of good and bad packets (CRC failures) received during the test

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*testRadioRxStats*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*testRadioRxStats*)
rxOk	INT16U		Number of packets successfully received
rxFailed	INT16U		Number of packets with error(s) received (CRC failures)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.20 getParameter<time>

Description

The *getParameter<time>* command may be used to request the current time on the mote. The mote reports time at the moment it is processing the command, so the information includes variable delay. For more precise time information consider using TIMEn pin (see *timeIndication*).

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*time*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*time*)
upTime	INT32U	none	Time from last reset (seconds)
utcTime	UTC_TIME_L	none	Long UTC time struct
asn	ASN	none	Time since start of network (slots)
asnOffset	INT16U	none	Time from the start of this slot (microseconds)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.6.21 getParameter<txPower>

Description

This command gets the radio output power in dBm, excluding any antenna gain. This value corresponds to the actual output power used by the radio driver and may not be the same as the input value entered with the setParameter<txPower>, which will set to nearest if the value entered is not supported by the hardware.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*txPower*)

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response code
paramId	INT8U	Parameters	Parameter ID (*txPower*)
txPower	INT8S	none	Transmit power, in dBm

Response Codes

Code	Description
RC_OK	Command completed successfully

5.7 getServiceInfo (0x12)

Description

The *getServiceInfo* command returns information about the service currently allocated to the mote.

Request

Parameter	Type	Enum	Description
destAddr	INT16U	none	Address of in-mesh destination of service. The manager address (0xFFFE) is the only value currently supported.
type	INT8U	Service Types	Service type (*bandwidth* or *latency*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
destAddr	INT16U	none	Short address of in-mesh service destination
type	INT8U	Service Types	Type of the service
state	INT8U	Service States	State of the service
value	INT32U	none	Inter-packet interval (in ms)

Response Codes

Code	Description
RC_OK	Command successfully completed

5.8 join (0x06)

Description

The *join* command requests that mote start searching for the network and attempt to join.The mote must be in the **Idle** state or the **Promiscuous Listen** state (see *search*) for this command to be valid. Note that the join time will be affected by the maximum current setting.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted
RC_INVALID_STATE	The mote is in invalid state to start join operation
RC_INCOMPLETE_JOIN_INFO	Incomplete configuration to start joining

5.9 lowPowerSleep (0x09)

Description

The *lowPowerSleep* command shuts down all peripherals and places the mote into deep sleep mode. The command executes after the mote sends its response. The mote enters deep sleep within two seconds after the command executes. The command may be issued at any time and will cause the mote to interrupt all in-progress network operation. To achieve a graceful disconnect, use the *disconnect* command before using the *lowPowerSleep* command. A hardware reset is required to bring a mote out of deep sleep mode.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted

5.10 openSocket (0x15)

Description

The *openSocket* command creates an endpoint for IP communication and returns an ID for the socket.

Request

Parameter	Type	Enum	Description
protocol	INT8U	Protocol Types	Protocol for this socket

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
socketId	INT8U		Socket ID

Response Codes

Code	Description
RC_OK	Command completed successfuly
RC_NO_RESOURCES	Couldn't create new socket due to resource availability

5.11 requestService (0x11)

Description

The *requestService* command may be used to request a new or changed service level to a destination device in the mesh. This command may only be used to update the service to a device with an existing connection (session).

Whenever a change in bandwidth assignment occurs, the application receives a *serviceChanged* event that it can use as a trigger to read the new service allocation.

Request

Parameter	Type	Enum	Description
destAddr	INT16U		Address of in-mesh destination of service. The manager address (0xFFFE) is the only value currently supported.
serviceType	INT8U	Service Types	Service type
value	INT32U		Inter-packet interval (in ms)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code

Response Codes

Code	Description
RC_OK	Service request accepted

5.12 reset (0x08)

Description

The *reset* command initiates a soft-reset of the device. The device will initiate the reset sequence shortly after sending out the response to this command. Resetting a mote directly can adversely impact its descendants; to disconnect gracefully from the network, use the *disconnect* command

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted

5.13 search (0x24)

Description

The *search* command requests that mote start listening for advertisements and report those heard from any network without attempting to join. This is called the **Promiscuous Listen** state. The mote must be in the **Idle** state for this command to be valid. The search state can be exited by issuing the *join* command or the *reset* command.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted
RC_INVALID_STATE	The mote is in invalid state to start search operation

5.14 sendTo (0x18)

Description

Send a packet into the network. If the command returns RC_OK, the mote has accepted the packet and has queued it up for transmission. A *txDone* notification will be issued when the packet has been sent, if and only if the packet ID passed in this command is different from 0xffff. You can set the packet ID to any value. The notification will contain the packet ID of the packet just sent, allowing association of the notification with a particular packet. The destination port should be in the range 0xF0B8-F0BF (61624-61631) to maximize payload.

Request

Parameter	Type	Enum	Description
socketId	INT8U	none	Socket ID
destIP	IPV6_ADDR	none	Destination IPV6 address. For applications not sending to an internet host (via an LBR), the manager well-known address (FF02::2) should be used.
destPort	INT16U	none	Destination port
serviceType	INT8U	Service Types	Service type.
priority	INT8U	Packet Priorities	Priority of the packet. Lower priority packets will queue behind higher ones. We recommend using medium priority for data traffic.
packetId	INT16U	none	User-defined packet ID for txDone notification; 0xFFFF=do not generate notification
payload	INT8U[]	none	Payload of the packet

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Packet was queued up for transmission

RC_NO_RESOURCES	No queue space to accept the packet

Payload Size Limits

Src, Dst Ports	To Manager	To Other Address	
Both are F0Bx	90	74	bytes
Either (or both) are F0xx	88	72	bytes
Any other	87	71	bytes

5.15 setParameter (0x01)

The *setParameter* command may be used to change parameters on the mote. The payload of each *setParameter* command begins with a parameter ID field that specifies the parameter being modified. A ll parameters modified with this command are nonvolatile and persist across reset and power cycle.

5.15.1 setParameter<advKey>

Description

Sets the Advertisement MIC key - this key is used to authenticate advertisements, and can be set per vendor/installation to prevent unauthorized devices from being able to respond to advertisements. If changed, it must match that set on the corresponding AP (using mset on the manager CLI) in order for the mote to join. It can be reset to default via the clearNV command.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (advKey)
advKey	SEC_KEY	none	16-byte Advertisement MIC key

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code
paramId	INT8U	Parameters	Parameter ID (advKey)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Couldn't update persistent storage

5.15.2 setParameter<antGain>

Description

The *setParameter<antGain>* command sets the antenna gain of the system (to properly calculate radiated power). Defaults to 2 dBi if not set.

Note: This parameter is available in devices running mote software >=1.4.x

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*antGain*)
antGain	INT8S	none	Antenna gain in dBi

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*antGain*)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Couldn't update persistent storage

5.15.3 setParameter<autoJoin>

Description

This command allows the microprocessor to change between automatic and manual joining by the mote's networking stack. In manual mode, an explicit *join* command from the application is required to initiate joining. This setting is persistent and takes effect after mote reset.

Note that auto join mode must not be set if the application is also configured to join (e.g combining 'auto join' with 'master' mode will result in mote not joining).

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (autoJoin)
mode	BOOL	none	0=manual join, 1=auto join

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (autoJoin)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Couldn't update persistent storage
RC_INVALID_VALUE	Invalid value specified for mode

5.15.4 setParameter<euCompliantMode>

Description

The *setParameter<euCompliantMode>* command may be used to set the EN 300 328 compliance mode that is used by the device. When enabled, the mote may skip some transmit opportunities to remain within average power limits. Motes below +10 dBm radiated power do not need to duty cycle to meet EN 300 328 requirements.

Note: This parameter is available in devices running mote software >=1.4.x

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*euCompliantMode*)
euCompliantMode	INT8U	none	EN 300 328 compliance mode. 0=off, 1=on

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*euCompMode*)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Couldn't update persistent storage

5.15.5 setParameter<eventMask>

Description

The *setParameter<eventMask>* command allows the microprocessor to selectively subscribe to event notifications. The default value of eventMask at mote reset is all 1s - all events are enabled. This setting is not persistent.

New event types may be added in future revisions of mote software. It is recommended that the client code only subscribe to known events and gracefully ignore all unknown events.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (eventMask)
eventMask	INT32U	Events	Event mask; 0=unsubscribe, 1=subscribe

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (eventMask)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.15.6 setParameter<joinDutyCycle>

Description

The *setParameter<joinDutyCycle>* command allows the microprocessor to control the ratio of active listen time to doze time (a low-power radio state) during the period when the mote is searching for the network. If you desire a faster join time at the risk of higher power consumption, use the *setParameter<joinDutyCycle>* command to increase the join duty cycle up to 100%. This setting is persistent and takes effect immediately if the device is searching for network.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*joinDutyCycle*)
dutyCycle	INT8U	none	Duty cycle (0-255), where 0=0.2% and 255=99.8%

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*joinDutyCycle*)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.15.7 setParameter<joinKey>

Description

The *setParameter<joinKey>* command may be used to set the join key in mote's persistent storage. Join keys are used by motes to establish secure connection with the network. The join key is used at next join.

Reading the *joinKey* parameter is prohibited for security reasons.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*joinKey*)
joinKey	SEC_KEY	none	Join key

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*joinKey*)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Could not write the key to storage

5.15.8 setParameter<macAddress>

Description

This command allows user to overwrite the manufacturer-assigned MAC address of the mote. The new value takes effect after the mote resets.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*macAddress*)
macAddress	MAC_ADDR	none	New MAC address to use

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*macAddress*)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Couldn't update persistent storage

5.15.9 setParameter<networkId>

Description

This command may be used to set the Network ID of the mote. This setting is persistent and is used on next join attempt.

As of version 1.4.x, a network ID of 0xFFFF can be used to indicate that the mote should join the first network heard.

> ⚠ 0xFFFF is never used over the air as a valid network ID - you should not set the Manager's network ID to 0xFFFF.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*networkId*)
networkId	INT16U	none	Network ID

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*networkId*)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Could not store the parameter

5.15.10 setParameter<OTAPLockout>

Description

This command allows the microprocessor to control whether Over-The-Air Programming (OTAP) of motes is allowed. This setting is persistent and takes effect immediately.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*OTAPLockout*)
mode	BOOL		0=otap allowed, 1=OTAP not allowed

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*OTAPLockout*)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_INVALID_VALUE	Invalid value of mode parameter
RC_WRITE_FAIL	Couldn't update persistent storage

5.15.11 setParameter<pwrSrcInfo>

Description

This command allows the microprocessor to configure power source information on the device. This setting is persistent and is used at network join time.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (powerSrcInfo)
maxStCurrent	INT16U		Maximum steady-state current, in µA; 0xFFFF=no limit
minLifetime	INT8U		Minimum lifetime, in months; 0=no limit
currentLimit_0	INT16U		Current limit, in µA; 0=limit entry empty
dischargePeriod_0	INT16U		Discharge period, in seconds
rechargePeriod_0	INT16U		Recharge period, in seconds
currentLimit_1	INT16U		Current limit, in µA; 0=limit entry empty
dischargePeriod_1	INT16U		Discharge period, in seconds
rechargePeriod_1	INT16U		Recharge period, in seconds
currentLimit_2	INT16U		Current limit, in µA; 0=limit entry empty
dischargePeriod_2	INT16U		Discharge period, in seconds
rechargePeriod_2	INT16U		Recharge period, in seconds

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (powerSrcInfo)

Response Codes

Code	Description
RC_OK	Command completed successfully

5.15.12 setParameter<routingMode>

Description

This command allows the microprocessor to control whether the mote will become a router once joined the network. If disabled, the manager will keep the mote a leaf node.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (routingMode)
mode	BOOL		0=routing, 1=non-routing

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (routingMode)

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_INVALID_VALUE	Invalid value of mode parameter

5.15.13 setParameter<txPower>

Description

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter ID (*txPower*)
txPower	INT8S	none	Transmit Power (dBm)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter ID (*txPower*)

Response Codes

Code	Description
RC_OK	Command completed sucessfuly
RC_INVALID_VALUE	The requested power is not supported

5.16 socketInfo (0x2B)

Description

Retrieve information about a socket. (Available in IP Mote >= 1.4.0)

Request

Parameter	Type	Enum	Description
index	INT8U	none	Index of requested socket. Indices start at 0

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
index	INT8U	none	The requested index
socketId	INT8U	none	The ID of the socket at the requested index
protocol	INT8U	none	Always 0 (UDP)
bindState	INT8U	none	0 = not bound, 1 = bound
port	INT16U	none	The port bound to this socket. 0 if not bound

Response Codes

Code	Description
RC_OK	Operation completed normally
RC_NOT_FOUND	A socket could not be found for the index provided

5.17 stopSearch (0x2F)

Description

The *stopSearch* command stops the search that was started either by the *join* or the *search* command. The mote must be in either **Promiscuous Listen** or **Search** state for this command to be valid. The mote goes back to **Idle** state when this command is received in a valid state. Available in mote >= 1.4.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted
RC_INVALID_STATE	The mote is in invalid state to stop search operation

5.18 testRadioRx (0x0C)

Description

The *testRadioRx* command clears all previously collected statistics and initiates a test of radio reception for the specified channel and duration. During the test, the mote keeps statistics about the number of packets received (with and without error). The test results may be retrieved using the *getParameter<testRadioRxStats>* command. The *testRadioRx* command may only be issued in **Idle** mode. The mote must be reset (either hardware or software reset) after radio tests are complete and prior to joining.

Station ID is available in IP mote >= 1.4, and WirelessHART mote >= 1.1.2. The station ID is a user selectable value used to isolate traffic if multiple tests are running in the same radio space. It must be set to match the station ID used by the transmitter.

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

Request

Parameter	Type	Enum	Description
channelMask	INT16U	none	Mask of channels (0–15) enabled for the test. Bit 0 corresponds to channel 0.Only one channel should be enabled.
time	INT16U	none	Test duration (in seconds).
stationId	INT8U	none	Unique (1-255) station id of this mote. 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 was accepted
RC_INVALID_VALUE	The mote is not in **Idle** state

RC_BUSY	Another test operation in progress

5.19 testRadioTxExt (0x28)

Description

The *testRadioTxExt* command allows the microprocessor to initiate a radio transmission test. This command may only be issued prior to the mote joining the network. Four types of transmission tests are supported:

- Packet transmission
- Continuous modulation
- Continuous wave (unmodulated signal)
- Packet transmission with clear channel assessment (CCA) enabled (Available in IP mote >= 1.4.0, and WirelessHART mote >= 1.1.2)

In a packet transmission test, the mote 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-3) 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 mote generates continuous pseudo-random modulated signal, centered at the specified channel. The test is stopped by resetting the mote.

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

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.

The *testRadioTxExt* command may only be issued when the mote is in **Idle** mode, prior to its joining the network. The mote must be reset (either hardware or software reset) after radio tests are complete and prior to joining.

Station ID is available in IP mote >= 1.4, and WirelessHART mote >= 1.1.2. The station ID is a user selectable value used in packet tests so that a receiver 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. See *testRadioRX* (SmartMesh IP) or *testRadioRxExt* (SmartMesh WirelessHART).

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

Request

Parameter	Type	Enum	Description
testType	INT8U	Radio Test Types	Type of transmission test
chanMask	INT16U	none	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	none	Number of times to repeat the packet sequence (0=do not stop). Applies only to packet transmission tests.
txPower	INT8S	none	Transmit power, in dB. Valid values are 0 and 8.
seqSize	INT8U	none	Number of packets in each sequence. This parameter is only used for packet test.
sequenceDef	seqDef[]	none	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 test. Each sequence definition is formatted as follows: INT8U pkLen; /* Length of packet (2-125 bytes) */ INT16U delay; /* Delay after packet transmission, microseconds */
stationId	INT8U	none	Unique (1-255) identifier included in packets. If this field is omitted, a stationId of 0 is used in the packet.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Command was accepted
RC_INVALID_STATE	Mote is not in correct state to accept the command
RC_BUSY	Mote is executing another radio test operation

5.20 zeroize (0x29)

Description

Zeroize (zeroise) command erases flash area that is used to store configuration parameters, such as join keys. This command is intended to satisfy zeroization requirement of FIPS-140 standard. After the command executes, the mote should be reset. Available in mote >= 1.4.x

> The zeroize command will render the mote inoperable. It must be re-programmed via SPI or JTAG in order to be useable.

Request

Parameter	Type	Enum	Description
password	INT32U	none	Password to prevent accidental erasure. The password is 57005 (0xDEAD).

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_ERASE_FAIL	Flash could not be erased due to unexpected internal error

6 Notifications

The following sections describe API payload of notifications supported by mote. Note that the API Header bytes are not listed.

API Header	API Payload

Notification Contents

6.1 advReceived

Description

The *advReceived* notification is generated by the mote w hen it is in **Promiscuous Listen** state (see the Mote States table) and it receives an advertisement.

Payload

Parameter	Type	Enum	Description
netId	INT16U	none	The Network ID contained in the advertisement just received
moteId	INT16U	none	The source address contained in the advertisement just received, i.e. the ID of its sender
rssi	INT8S	none	The Received Signal Strength Indicator (RSSI) at which the advertisement was received
joinPri	INT8U	none	The join priority (hop depth) contained in the advertisement just received

6.2 events

Description

The mote sends an *events* notification to inform the application of new events that occurred since the previous *events* notification. The notification also contains up-to-date information about current mote state and any pending alarms.

Payload

Parameter	Type	Enum	Description
events	INT32U	Events	Bitmap of recent events
state	INT8U	Mote States	Current mote state
alarmsList	INT32U	Alarms	Bitmap of current alarms

6.3 receive

Description

Informs the application that a packet was received.

Payload

Parameter	Type	Enum	Description
socketId	INT8U	none	Socket ID on which the packet was received
srcAddr	IPV6_ADDR	none	Source address of the packet
srcPort	INT16U	none	Source port of the packet
payload	INT8U[]	none	Payload of the packet

6.4 timeIndication

Description

The *timeIndication* notification applies to mote products that support a time interrupt into the mote. The time packet includes the network time and the current UTC time relative to the manager.

For LTC5800-IPM based products, driving the TIMEn pin low (assert) wakes the processor. The pin must asserted for a minimum of t_{strobe} µs. De-asserting the pin latches the time, and a *timeIndication* will be generated within $t_{response}$ ms. Refer to the LTC5800-IPM Datasheet for additional information about TIME pin usage.

> ⚠ The processor will remain awake and drawing current while the TIMEn pin is asserted. To avoid drawing excess current, take care to minimize the duration of the TIMEn pin being asserted past t_{strobe} minimum.

Payload

Parameter	Type	Enum	Description
uptime	INT32U	none	Time from last reset (seconds)
utcTime	UTC_TIME_L	none	UTC time at TIMEn trigger
asn	ASN	none	Time, in an integer number of slots, at TIMEn trigger
asnOffset	INT16U	none	Time from the start of this slot in microseconds
asnSubOffset	INT16U	none	Time from the start of reported microsecond, in nanoseconds. On LTC5800-IPM, granularity of this field is 250ns. This field is available in mote version >= 1.4.0

6.5 txDone

Description

The *txDone* notification informs the application that the mote has finished sending a packet. This notification will only be generated if the user has provided a valid (0x0000-0xFFFE) packet ID when calling the *sendTo* command.

Payload

Parameter	Type	Enum	Description
packetId	INT16U	none	Packet id provided in *sendTo* call
status	INT8U	Transmit Status	Packet transmission status

7 Definitions

7.1 Commands

The following two tables provides a summary of mote API commands and notifications corresponding to the command ID field in the API header. For correct operation, all packets must be acknowledged. Any unrecognized command or notification should be acknowledged with an RC_UNKNOWN_CMD response code.

Name	Value	Description
setParameter	0x01	Set Parameter
getParameter	0x02	Get Parameter
join	0x06	Join the network
disconnect	0x07	Disconnect from the network
reset	0x08	Reset mote
lowPowerSleep	0x09	Low power sleep
testRadioRx	0x0C	Start receive radio test
clearNV	0x10	Clear non-volatile configuration storage
requestService	0x11	Request service
getServiceInfo	0x12	Get service information
openSocket	0x15	Open UDP socket
closeSocket	0x16	Close UDP socket
bindSocket	0x17	Bind UDP socket
sendTo	0x18	Send packet
search	0x24	Go into promiscuous listen mode
testRadioTxExt	0x28	Start transmit radio test
zeroize	0x29	Erase all mote configuration
socketInfo	0x2B	Read the binding state of a socket
blinkPayload	0x2E	Write blink payload

stopSearch	0x2F	Stop search (added in IP Stack 1.4)

7.2 Notifications

The following notification types may be generated by the mote:

Name	Value	Description
timeIndication	0x0D	Time information
events	0x0F	Event(s)
receive	0x19	Packet received
txDone	0x25	Transmit done
advReceived	0x26	Received advertisement (promiscuous listen mode)

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. 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.

7.3 Parameters

The following parameters may be used in *setParameter* and *getParameter* commands

Name	Value	Persistent	Description
macAddress	0x01	Y	MAC address of the mote
joinKey	0x02	Y	Security join key
networkId	0x03	Y	Network identifier
txPower	0x04	Y	Transmit power
joinDutyCycle	0x06	Y	Join duty cycle
eventMask	0x0B		Event mask
moteInfo	0x0C		Mote information
netInfo	0x0D		Network information
moteStatus	0x0E		Mote status
time	0x0F		Time information
charge	0x10		Charge
testRadioRxStats	0x11		Statistics from radio test
OTAPLockout	0x15	Y	OTAP lockout
moteId	0x17		MoteId, a.k.a mote's short address
ipv6Address	0x18		Mote's IPv6 address
routingMode	0x1D	Y	Routing mode
appInfo	0x1E	Y	Application Information
powerSrcInfo	0x1F	Y	Power Source Information
advKey	0x22	Y	Advertising MIC key
autoJoin	0x24	Y	Auto Join mode
antGain	0x29	Y	Antenna gain (default 2 dBi)
euCompliantMode	0x2A	Y	EN 300 328 compliance
sizeInfoExt	0x2B		Extended size info

entropy	0x2C		Entropy (added in IP Stack 1.4)

7.4 Response Codes

The following response codes are valid for the API. Refer to the individual command documentation for specific reasons for response codes.

Name	Value	Description
RC_OK	0x00	Command completed successfully
RC_ERROR	0x01	Processing error
RC_BUSY	0x03	Device currently unavailable to perform the operation
RC_INVALID_LEN	0x04	Invalid length
RC_INVALID_STATE	0x05	Invalid state
RC_UNSUPPORTED	0x06	Unsupported command or operation
RC_UNKNOWN_PARAM	0x07	Unknown parameter
RC_UNKNOWN_CMD	0x08	Unknown command
RC_WRITE_FAIL	0x09	Couldn't write to persistent storage
RC_READ_FAIL	0x0A	Couldn't read from persistent storage
RC_LOW_VOLTAGE	0x0B	Low voltage detected
RC_NO_RESOURCES	0x0C	Couldn't process command due to low resources (e.g. no buffers)
RC_INCOMPLETE_JOIN_INFO	0x0D	Incomplete configuration to start joining
RC_NOT_FOUND	0x0E	Resource not found
RC_INVALID_VALUE	0x0F	Invalid value supplied
RC_ACCESS_DENIED	0x10	Access to resource or command is denied
RC_ERASE_FAIL	0x12	Erase operation failed

7.5 Service Types

The following table lists types of services.

Name	Value	Description
bandwidth	0x00	Bandwidth-type service

7.6 Service States

The following table lists state that a service may be found in.

Name	Value	Description
completed	0x00	Service request completed (idle)
pending	0x01	Service request pending

7.7 Protocol Types

The following protocol may be used to send and receive packets.

Name	Value	Description
udp	0x00	User Datagram Protocol (UDP)

7.8 Packet Priorities

The following table lists priorities that may be used to send packets.

Name	Value	Description
low	0x00	Low priority
medium	0x01	Medium priority
high	0x02	High priority

7.9 Mote States

The following table lists mote states

Name	Value	Description
init	0x00	Initializing
idle	0x01	Idle, ready to be configured or join
searching	0x02	Searching for network
negotiating	0x03	Sent join request

connected	0x04	Received at least one packet from the Manager
operational	0x05	Configured by Manager and ready to send data
disconnected	0x06	Disconnected from the network
radiotest	0x07	The mote is in radio test mode
promiscuous listen	0x08	The mote received *search* command and is in promiscuous listen mode
blink	0x09	The mote received a Blink command and will stay for 60s without any MAC-layer acks

7.10 Events

Note: multiple events may be reported in a single event notification

Name	Value	Description
boot	0x0001	The mote booted up
alarmChange	0x0002	Alarm(s) were opened or closed
timeChange	0x0004	UTC time-mapping on the mote changed
joinFail	0x0008	Join operation failed
disconnected	0x0010	The mote disconnected from the network
operational	0x0020	Mote has connection to the network and is ready to send data
svcChange	0x0080	Service allocation has changed
joinStarted	0x0100	Mote started joining the network

7.11 Alarms

The mote may declare the following alarms:

Name	Value	Description
nvError	0x01	Detected an error in persistent configuration storage (NV)
otpError	0x04	Detected an error in calibration or bsp data in flash
notReady	0x08	Mote can't forward data packets to the network (usually due to congestion)

7.12 Radio Test Types

The following tests may be performed via *testRadioTx* command:

Name	Value	Description
packet	0x00	Packet transmission
cm	0x01	Continuous modulation
cw	0x02	Continuous wave
pkcca	0x03	Packet test with clear channel assesment

7.13 Packet Transmit Status

Packet transmit status is returned as part of *txDone* notification.

Name	Value	Description
ok	0x00	Packet sent into the network
fail	0x01	Packet dropped

Trademarks

Eterna, Mote-on-Chip, and SmartMesh IP, are trademarks of Dust Networks, Inc. The Dust Networks logo, Dust, Dust Networks, and SmartMesh are registered trademarks of Dust Networks, Inc. LT, LTC, LTM and  are registered trademarks of Linear Technology Corp. All third-party brand and product names are the trademarks of their respective owners and are used solely for informational purposes.

Copyright

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.

Linear Technology 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.

Linear Technology 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. Linear Technology customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify and hold Linear Technology 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 Linear Technology was negligent regarding the design or manufacture of its products.

Linear Technology 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.

Linear Technology does not warrant or represent that any license, either express or implied, is granted under any Linear Technology patent right, copyright, mask work right, or other Linear Technology intellectual property right relating to any combination, machine, or process in which Linear Technology products or services are used. Information published by Linear Technology regarding third-party products or services does not constitute a license from Linear Technology 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 Linear Technology under the patents or other intellectual property of Linear Technology.

Dust Networks, Inc is a wholly owned subsidiary of Linear Technology Corporation.