SmartMesh WirelessHART Mote Serial API Guide

Table of Contents

1 About This Guide

1.1 Related Documents

The following documents are available for the SmartMesh WirelessHART network:

Getting Started with a Starter Kit

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

User Guide

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

Interfaces for Interaction with a Device

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

Software Development Tools

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

Application Notes

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

Documents Useful When Starting a New Design

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

Other Useful Documents

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

1.2 Conventions Used

The following conventions are used in this document:

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

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

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

Tips provide useful information about the product.

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	07/12/2012	Initial Release
2	08/10/2012	Added legacy and extended radiotest APIs
3	03/18/2013	Numerous small changes
4	10/22/2013	Document retitled. Added lock and compliantMode APIs; Updated getParameter<moteInfo>; Clarification of send request/response use of RC code.
5	04/04/2014	Updated and clarified radiotest commands;
6	10/28/2014	Included command IDs in titles; Added Station ID to testRadio commands; Added notReady alarm to Definitions; Clarified Notification structure; Clarified packet format; Clarified use of Packet ID; Added description of autojoin; Updated SetParameter<hartDeviceStatus> command
7	01/21/2015	Fixed title and incorrect parameters for the getNVParameter<lock> command; Other small corrections
8	04/22/2015	Clarified autojoin version requirements; Fixed events table; Other small corrections
9	12/03/2015	Added euCompliantMode and hartCompliantMode parameters; Other small corrections
10	12/11/2015	Documented broadcast transport bit in API flags
11	01/29/2016	Added radio test type for CCA
12	11/07/2016	Added joinShedTime parameter

2 Introduction

This guide describes the commands used by an external processor to communicate with the SmartMesh WirelessHART 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 WirelessHART 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.

LTP5900-WHM only:
The API UART on the LTP5900-WHM supports 2 legacy modes, selected via the Mode B pin (pin 11):

- Mode 1: 9600 baud, HDLC encoded
  - 3/4/5-wire interface - see the datasheet
  - Mode B pin low
- Mode 3: 115.2K baud, HDLC encoded
  - 5-wire interface - see the datasheet
  - Mode B pin high

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

The fuse table for the LTC5800-WHM 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-WHM, or the Starter Kit mote (DC9003A-C), the fuse table has been pre-programmed for mode 4 at 115.2 Kbps and cannot be changed.

4 Protocol

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

In addition, HART defines an additional time data type

Type	Length (bytes)	Notes
HART_TIME	4	HART time in 1/32 of a ms.

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 Maximum packet size

Both request and response packets have a maximum payload size of 128 bytes before HDLC encoding. This does not include subsequent addition of start/stop delimiters, frame checksum, or octet-stuffing escape sequences.

4.2.2 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**

4.2.3 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 Command Identifiers)
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	Ignore Packet ID; 0 = do not ignore, 1 = ignore
3	Sync
4-7 (MSB)	Command-specific flags

Request/Response flag

The Request/Response flag identifies a packet as containing a Serial API Request or Response payload. The payload of each command and notification is unique and is defined in the sections below. Note that this is not to be confused with end-to-end Transport direction flags for *send* and *dataReceived* (bit 7).

Packet ID

Packet ID is a 1 bit sequence number that is used to ensure reliable, in-order processing of packets. When the mote boots it may start the first packet with ID=0 or ID=1 and the sync bit set, i.e. the flags are either 0x08 or 0x0A .

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 should be processed and the response should contain a copy of packet ID. If a duplicate packet ID is received, a cached copy of response should be 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.

Ignore Packet ID

The packet ID feature improves the reliability of the serial interface between the mote and the microprocessor and is therefore highly recommended. However, the OEM microprocessor does have the option of disabling packet ID toggling to simplify coding at the increased risk of duplicate packets due to retransmissions across the serial interface. Specifically, the OEM microprocessor may elect to ignore packet IDs during one or both of the following:

- Requests from the microprocessor — When sending requests to the mote, the microprocessor may disable packet ID toggling (in both the request and response) by setting the “Ignore packet ID” bit in the flags byte of the request. In this case, the mote cannot discern duplicate serial packets and thus will process all serial packets it receives.
- Requests from mote — The mote will always send requests to the microprocessor with packet ID toggling enabled. The OEM microprocessor should maintain an internal packet ID to discern duplicate packets from the mote. However, if the application tolerates the possibility of duplicate packets due to serial retries, the OEM microprocessor may ignore the packet ID of the incoming request. In this case, the OEM microprocessor should acknowledge the request with the “Ignore packet ID” bit in the flags byte set to ignore.

You must either always use packet id or never use it - alternating between using and not using it may result in communications errors.

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.

Command-specific flags

The following bits have special meanings in the context of separate commands:

setNvParameter command

Bit	Description
7	0 = set NV only, 1 = set NV & RAM

send command and dataReceived notification

Bit	Description
5	Transport session: 0 = unicast, 1 = broadcast
6	Transport type: 0 = best effort, 1 = acknowledged
7	Transport direction: 0 = end-to-end request, 1 = end-to-end response

A *send* command issued in response to an acknowledged *dataReceived* notification (bit 6 set) should set the transport type to "acknowledged" (bit 6) and set the transport direction to "end-to-end response" (bit 7) and match the transport session (bit 5) to the value from the *dataReceived* notification. To publish data, a *send* should set the transport type to "best effort" and the transport direction to "end-to-end response."

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.

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 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 reported via *getParameter<moteInfo>* will be changed.

5 Commands

5.1 clearNV (0x10)

Description

The *clearNV* command resets the mote's Non-Volatile (NV) memory to its factory-default state. Refer to the WirelessHART User Guide for table of default values. Note that since this command clears the mote's security join counter, the corresponding manager's Access Control List (ACL) entry may need to be cleared as well to allow joining.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Flash operation failed
RC_LOW_VOLTAGE	Low voltage

5.2 disconnect (0x07)

Description

The *disconnect* command requests that the mote disconnect from the network. The mote will send an indication to its network neighbors that it is about to become unavailable. Just after the mote disconnects, it sends the microprocessor an events packet with the disconnected bit set, indicating it will reset. This command is only valid in when the mote is in the **Connected** or **Operational** state (see Mote State).

The OEM microprocessor should disconnect from the network if the device is going to power down, reset, or otherwise be unavailable for a long period.

> ⚠ A mote will reset itself after having sent the *disconnect* notification to the OEM microprocessor. The microprocessor should wait to acknowledge the *boot* event before shutting down.

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

5.3 file<open> (0x19)

Description

The *fileOpen* command may be used to open the scratchpad file in the mote filesystem.

Request

Parameter	Type	Enum	Description
name	INT8U[12]	none	Name of the file. Only "3user.dat" is currently permitted.
options	INT8U	File Open Options	File open options. Must use "create" option.
size	INT16U	none	File size. Up to 2048 bytes.
mode	INT8U	File Open Mode	File open mode. Recommend opening read/write, but must use shadow in addition.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Return code
descriptor	INT32	none	File descriptor of the opened file (only valid if rc=0)

Response Codes

Code	Description
RC_OK	Command was accepted
RC_OPEN_FAIL	File could not be opened
RC_INVALID_VALUE	Invalid value for a parameter

5.4 file<read> (0x18)

Description

The *fileRead* command may be used to read data stored in the scratchpad file in the mote filesystem. The size of the data read is limited by the size of a serial API transaction.

Request

Parameter	Type	Enum	Description
descriptor	INT32	none	The file descriptor returned in *fileOpen*.
offset	INT16U	none	Offset from start of file where data is to be read
length	INT8U	none	Length of data to be read, in bytes

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code
descriptor	INT32	none	File descriptor of file read
offset	INT16U	none	Offset from start of file where data was read
length	INT8U	none	Length of data read, in bytes
data	INT8U[]	none	The read data

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_READ_FAIL	File could not be read
RC_INVALID_VALUE	Invalid value for a parameter
RC_NOT_FOUND	File descriptor was not found

5.5 file<write> (0x17)

Description

The *fileWrite* command may be used to read data stored in the scratchpad file in the mote filesystem. The size of the data read is limited by the size of a serial API transaction.

Request

Parameter	Type	Enum	Description
descriptor	INT32	none	The file descriptor returned in *fileOpen*.
offset	INT16U	none	Offset from start of file where data is to be written
length	INT8U	none	Length of data to be written, in bytes
data	INT8U[]	none	The data to be written

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
length	INT32	none	Length of the written data

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_WRITE_FAIL	File could not be written
RC_INVALID_VALUE	Invalid value for a parameter
RC_NOT_FOUND	File descriptor was not found

5.6 getParameter (0x02)

The g*etParameter* command may be used to retrieve current settings on the mote. The payload of each g*etParameter* commands begins with a parameter Id field that specifies the parameter being accessed.

5.6.1 getParameter<joinDutyCycle>

Description

The getParameter<joinDutyCycle> command return mote's join duty cycle, which determines the percentage of time the mote spends in radio receive mode while searching for network. The value of join duty cycle is expressed in increments of 1/255th of 100%, where 0 corresponds to 0% and 255 corresponds to 100%.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*joinDutyCycle*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Result code
paramId	INT8U	Parameter IDs	Parameter identifier (*joinDutyCycle*)
joinDutyCycle	INT8U	none	Duty cycle (0-255), where 0=0% and 255=100%

Response Codes

Code	Description
RC_OK	Operation was successfully completed

5.6.2 getParameter<service>

Description

The *getParameter<service>* command retrieves information about the service allocation that is currently available to the field device. Services (now called "Timetables" in WirelessHART 7.4) in the range 0x00-7F are those requested by the device, and those in the range 0x80-FF are assigned independently by the network manager.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*service*)
serviceId	INT8U	none	Service identifier. 0x00-0xFF

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*service*)
serviceId	INT8U	none	Service identifier. 0x00-0xFF
serviceState	INT8U	Service State	Service state
serviceFlags	INT8U	Service Flags	Service flags
appDomain	INT8U	Application Domain	Application domain
destAddr	INT16U	none	Destination address
time	INT32U	none	Time value (period or latency)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length

RC_UNKNOWN_PARAM	Unknown parameter value
RC_NOT_FOUND	Service not found

5.6.3 getParameter<moteInfo>

Description

The *getParameter<moteInfo>* command returns static information about the mote's hardware and software. Note that network state-related information about the mote may be retrieved using *getParameter<networkInfo>*.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*moteInfo*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Result code
paramId	INT8U	Parameter IDs	Parameter identifier (*moteInfo*)
apiVersion	INT8U	none	API version
serialNum	INT8U[8]	none	Serial number
hwModel	INT8U	none	Hardware model
hwRev	INT8U	none	Hardware revision
swMajorRev	INT8U	none	Software major revision
swMinorRev	INT8U	none	Software minor revision
swPatch	INT8U	none	Software patch number
swBuild	INT16U	none	Software build number

Response Codes

Code	Description
RC_OK	Operation was successfully completed

RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.6.4 getParameter<networkInfo>

Description

The *getParameter<networkInfo>* command may be used to retrieve the mote's network-related parameters. Note that static information about the mote's hardware and software may be retrieved using *getParameter<moteInfo>*.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*networkInfo*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*networkInfo*)
macAddress	MAC_ADDR	none	The MAC address is the vendor IEEE address used by the mote. This value defaults to the factory-configured serial number, unless overwritten via *setNVParameter<macAddress>* command.
moteId	INT16U	none	Mote ID is a short address assigned to the mote by the manager at network join time
networkId	INT16U	none	Network ID

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.6.5 getParameter<moteStatus>

Description

The *getParameter<moteStatus>* command is used to retrieve the mote's state and frequently changing information. Note that static information about the state of the mote hardware and software may be retrieved using *getParameter<moteInfo>*.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*moteStatus*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*moteStatus*)
state	INT8U	Mote State	Mote state
moteStateReason	INT8U	none	Reserved field
changeCounter	INT16U	none	Change counter
numParents	INT8U	none	Number of parents
moteAlarms	INT32U	Mote Alarms	Current mote alarms
statusFlags	INT8U	Status Flags	Status flags

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.6.6 getParameter<time>

Description

The *getParameter<time>* command is used to request the current time on the mote.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*time*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*time*)
utcTime	UTC_TIME	none	UTC time. The time is propagated from the manager, and therefore may depend on an external time source, such as NTP.
asn	ASN	none	ASN is the absolute slot number-the number of timeslots since the access point last reset
asnOffset	INT16U	none	The ASN offset is the number of microseconds since the beginning of the current slot

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.6.7 getParameter<charge>

Description

The *getParameter<charge>* command retrieves estimated charge consumption of the mote since the last reset, as well as the mote uptime and last measured temperature.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*charge*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*charge*)
charge	INT32U	none	Charge since last reset (mC)
uptime	INT32U	none	Uptime since last reset (sec)
temperature	INT8S	none	Temperature (°C)
fractionalTemp	INT8U	none	Temperature in 1/255 of °C

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.6.8 getParameter<testRadioRxStats>

Description

The *getParameter<testRadioRxStats>* command retrieves statistics for the latest radio reception 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	Parameter IDs	Parameter identifier (*testRadioRxStats*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*testRadioRxStats*)
rxOk	INT16U	none	Number of packets successfully received
rxFailed	INT16U	none	Number of packets with error(s) received (CRC failures)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value.

5.6.9 getParameter<lock>

Description

The *getParameter<lock>* command returns the current (RAM resident) lock code and locking master. To determine what the lock status will be after reset, use the *getNVParameter<lock>* command. Note: This parameter is available in devices running mote software >= 1.1.0

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter Identifier (*lock*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
parameterId	INT8U	Parameter IDs	Parameter identifier (*lock*)
code	INT8U	none	One of 0=unlock, 1=lock_master, 2=lock_all
master	INT16U	none	Short address of locking master (F981 for GW, 0000 for serial)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.7 getNVParameter (0x04)

The *getNVParameter* command can be used to retrieve the value of persistent parameters on the mote.

5.7.1 getNVParameter<autojoin>

Description

The *getNVParameter<autojoin>* command returns the autojoin status stored in mote's persistent storage (i.e. set with *setNVParameter<autojoin>*). Autojoin can be used to cause a mote in slave mode to join on its own when booted.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*autojoin*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*autojoin*)
autojoin	INT8U	none	0 = off, 1 = on

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.2 getNVParameter<euCompliantMode>

Description

The *getNVParameter<euCompliantMode>* command may be used to retrieve the EN 300 328 compliance mode that is used by devices. 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 >=1.2.x

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (euCompliantMode)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (hartCompliantMode)
euCompliantMode	INT8U	none	EN 300 328 compliance mode. 0=off, 1=on

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete

5.7.3 getNVParameter<HARTantennaGain>

Description

The *getNVParameter<HARTantennaGain>* command reads the antenna gain value from the mote's persistent storage. This value is added to conducted output power of the Dust mote when replying to HART command 797 (Write Radio Power Output) and to HART command 798 (Read Radio Output Power).

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*HARTantennaGain*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*HARTantennaGain*)
antennaGain	INT8S	none	Antenna gain in dBi

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.4 getNVParameter<hartCompliantMode>

Description

The *getNVParameter<hartCompliantMode>* command may be used to retrieve the HART compliance mode that is used by devices. This mode controls strict compliance to HART specification requirements, specifically:

- join timeouts (faster in non-compliant mode)
- Keepalive interval (adapts to synch quality in non-compliant mode)
- Health report format (uses saturating counters in non-compliant mode)

Note: This parameter is available in devices running mote software >= 1.1.0

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (hartCompliantMode)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (hartCompliantMode)
hartCompliantMode	INT8U	HART Compliance Mode	Compliance mode. 0=OFF (default), 1=ON

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete

5.7.5 getNVParameter<joinShedTime>

Description

The *getNVParameter<joinShedTime>* command returns the join shed time used with HART command 771/772 to determine when the mote should transition between active and passive search.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*joinShedTime*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*joinShedTime*)
joinShedTime	INT32U	HART_TIME	Active search shed time. See HART spec 155 command 771/772

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.6 getNVParameter<lock>

Description

The *getNVParameter<lock>* command returns the persisted lock code and locking master (those to be used after reset). To determine the current lock status, use the *getParameter<lock>* command. Note: This parameter is available in devices running mote software >= 1.1.0

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*lock*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*lock*)
code	INT8U	none	One of 0=unlock, 1=lock_master, 2=lock_all
master	INT16U	none	Short address of locking master (F981 for GW, 0000 for serial)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.7.7 getNVParameter<macAddress>

Description

The *getNVParameter<macAddress>* command returns the MAC address stored in mote's persistent storage (i.e. set with *setNVParameter<macAddress>*).

> ⚠ This command returns 0000000000000000 if the *macAddress* has not been set previously - the mote will use its hardware MAC in this case, but it is not displayed with this command.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*macAddress*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*macAddress*)
macAddr	MAC_ADDR	none	MAC address of the device

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.8 getNVParameter<networkId>

Description

The *getNVParameter<networkId>* command returns the Network ID stored in mote's persistent storage.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*networkId*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*networkId*)
networkId	INT16U	none	Network ID

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.9 getNVParameter<OTAPlockout>

Description

The *getNVParameter<OTAPlockout>* command reads the OTAP lockout setting from the mote's persistent storage. OTAP lockout specifies whether the mote can be Over-The-Air-Programmed (OTAP).

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*OTAPlockout*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*OTAPlockout*)
otapLockout	INT8U	OTAP Lockout	OTAP lockout setting

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.10 getNVParameter<powerInfo>

Description

The *getNVParameter<powerInfo>* command returns the power supply information stored in mote's persistent storage.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*powerInfo*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*powerInfo*)
powerSource	INT8U	Power Source	Power source
dischargeCur	INT16U	none	Discharge current (µA). The discharge current field indicates the maximum average current available to the mote. The manager will assign links such that this maximum average will not be exceeded, but may assign much lower as required by topology and data rates. This may limit the function of the device. For example, high-speed pipes may not be available if their use would exceed the discharge current.
dischargeTime	INT32U	none	Discharge time (sec). The discharge time field specifies the duration that the power source can supply the discharge current. In the case where the device is line-powered, or has a battery that will last longer than 232 * 1/32 ms (approximately 37 hours), the discharge time will not be used in link assignment and should be set to 0xFFFF FFFF.
recoverTime	INT32U	none	Recover time (sec). The recover time field specifies the time the device power supply takes to recharge under no load before the discharge current is available. If the power source is not rechargeable, recover time should be set to zero.

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.11 getNVParameter<ttl>

Description

The *getNVParameter<ttl>* command reads the Time To Live parameter from the mote's persistent storage. Time To Live is used when the mote sends a packet into the network, and specifies the maximum number of hops the packet may traverse before it is discarded from the network.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*ttl*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*ttl*)
timeToLive	INT8U	none	Time To Live

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.7.12 getNVParameter<txPower>

Description

The *getNVParameter<txPower>* command returns the transmit power value stored in mote's persistent storage.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*txPower*)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*txPower*)
txPower	INT8S	none	Transmit power, in dBm

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_READ_FAIL	Read did not complete
RC_LOW_VOLTAGE	Voltage check failed

5.8 hartPayload (0x0A)

Description

The *hartPayload* command allows the microprocessor to forward a HART payload to the mote. The format of the command must be as follows

```
16-bit command number | data length | data
```

The reply (if any) will be in the form of a HART response and sent in the payload of the acknowledgement. The RC_INVALID_VALUE response means that the *hartPayload* command was given a HART command that the mote does not terminate.

Request

Parameter	Type	Enum	Description
payloadLen	INT8U	none	Payload length
payload	INT8U[]	none	Payload

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
payloadLen	INT8U	none	Payload length
payload	INT8U[]	none	Payload

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_INVALID_VALUE	Invalid value

5.9 join (0x06)

Description

The *join* command requests that a 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. The join time is partly determined by the join duty cycle. For guidance on setting the join duty cycle, see *setParameter<joinDutyCycle>*.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_STATE	Invalid state for join
RC_INCOMPLETE_JOIN_INFO	Incomplete join information

5.10 lowPowerSleep (0x09)

Description

The *lowPowerSleep* command shuts down all peripherals and places the mote in deep sleep mode. The *lowPowerSleep* command may be issued at any time and will cause the mote to interrupt all in-progress network operation. The command executes after the mote sends its response. The mote enters deep sleep within two seconds after the command executes.

The OEM microprocessor should put the mote into low power sleep when the mote needs to be offline for an extended period of time. In most cases, this will result in a lower current state of the mote than simply asserting /RST without putting the mote in low power sleep. To achieve a graceful disconnect, use the *disconnect* command before using the *lowPowerSleep* command. The mote can only be awakened from low power sleep by asserting a non-maskable interrupt, such as the /RST control line. For power consumption information, refer to the mote product datasheet.

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

5.11 reset (0x08)

Description

Upon receiving this command, the mote resets itself after a short delay. The mote will always send a response packet before initiating the reset. To force the mote to gracefully leave 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

5.12 search (0x11)

Description

The *search* command causes the mote to listen for network advertisements and notify the microprocessor about each advertisement it hears. This is referred to as the **Promiscuous Listen** state. Notifications are sent using the *advReceived* notification. The *search* command may only be issued prior to join. The mote stays in listen mode until the *join* command is received or the mote is reset.

Request

Parameter	Type	Enum	Description

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_STATE	Mote is in invalid state for this operation

5.13 send (0x05)

Description

The *send* command allows a serial device to send a packet into the network through the mote's serial port. The mote forwards the packet to the network upon receiving it. The microprocessor must not attempt to send data at a rate that exceeds its allocated bandwidth. For a WirelessHART device, the payload of the packet must include the status byte and the extended status byte, followed by one or more sets of HART commands up to the maximum *send* payload size, as follows:

```
Request: Status|Extended Status|Cmd1|Length1|Data1|Cmd2|Length2|Data2...
Response: Status|Extended Status|Cmd1|Length1(includes response
code)|RC1|Data1|Cmd2|Length2|RC2|Data2...
```

Prior to sending the payload into the network, the mote caches the value of Status and Extended Status to use in packets it originates locally. The *send* command is only valid when the mote is in the **Operational** state. If the mote receives this command when it is not in the **Operational** state, it returns the error RC_INV_STATE. Note: The serial device can receive a request while the mote is in the process of transition from the **Connected** state to the **Operational** state.

Request

Parameter	Type	Enum	Description
destAddr	INT16U	none	Destination address. If the send command is a response to a *dataReceived* command request, the destination address should be set to the source address from the *dataReceived* command. Otherwise, use 0xF981 for the destination address.
serviceId	INT8U	none	Service ID. The service ID must specify a valid service.
appDomain	INT8U	Application Domain	Application domain. For an end-to-end response to a reliable *dataReceived* command request, the application domain should be set to “maintenance.”
priority	INT8U	Priority	The mote maintains a queue of packets to be sent into the network. The queue is sorted in order of priority.
reserved	INT16U	none	Reserved field, set to 0xFFFF
seqNum	INT8U	none	Sequence number. The sequence number is required when sending an end-to-end reliable response packet. This number must be the sequence number of the end-to-end request packet previously received via the *dataReceived* command. When sending a reliable request, the mote manages the sequence number on behalf of the network device. Therefore, if sending a reliable request or an unreliable packet use the value 0xFF for the sequence number.

payloadLen	INT8U	none	Payload length. The maximum length of the message payload is 94 bytes.
payload	INT8U[]	none	Payload. The format of the serial packet payload is transparent to the mote. For non HART-compliant applications, messages must be prepended with a 4-byte header, 00 00 FC 12. This corresponds to the Status, Extended Status, and 2-byte command fields.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_CMD	Unknown command
RC_INVALID_VALUE	Invalid application domain or priority
RC_NOT_FOUND	Service or route to destination not found
RC_NO_RESOURCES	Mote buffers are full
RC_INVALID_STATE	Mote is not in **Operational** state

5.14 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 volatile do not persist across multiple resets.

5.14.1 setParameter<txPower>

Description

The *setParameter<txPower>* command sets the mote conducted RF output power. Refer to product datasheets for supported RF output power values. For example, if the mote has a typical RF output power of +8 dBm when the Power Amplifier (PA) is enabled, set the *txPower* parameter to 8 to enable the PA. Similarly, if the mote has a typical RF output power of -2 dBm when the PA is disabled, then set the *txPower* parameter to -2 to turn off the PA. Note that this value is the RF output power coming out of the mote and not the radiated power coming out of the antenna. This command may be issued at any time and takes effect upon the next transmission.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*txPower*)
txPower	INT8S	none	Transmit power, in dBm

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*txPower*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value
RC_INVALID_VALUE	Invalid value

5.14.2 setParameter<joinDutyCycle>

Description

The *setParameter<joinDutyCycle>* command allows the microprocessor to control the join duty cycle – 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. The default duty cycle enables the mote to join the network at a reasonable rate without using excessive battery power. If you desire a faster join time at the cost of higher power consumption, use the *setParameter<joinDutyCycle>* command to increase the join duty cycle up to 100%. Note that the *setParameter<joinDutyCycle>* command is not persistent and stays in effect only until reset. For power consumption information, refer to the mote product datasheet.

This command may be issued multiple times during the joining process. This command is only effective when the mote is in the **Idle** and **Searching** states.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*joinDutyCycle*)
dutyCycle	INT8U	none	Duty cycle (0-255), where 0=0% and 255=100%

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*joinDutyCycle*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.14.3 setParameter<batteryLife>

Description

The *setParameter<batteryLife>* command allows the microprocessor to update the remaining battery life information that the mote reports to WirelessHART Gateway in Command 778. This parameter must be set during the **Idle** state prior to joining, and should be updated periodically throughout operation. This parameter is only used in WirelessHART-compliant devices.

> ⚠ Command 778 is deprecated in version 7.4 of the HART specification as most existing gateways do not use battery life information.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*batteryLife*)
batteryLife	INT16U	none	Estimated remaining battery life (days)
powerStatus	INT8U	Power Status	Power status

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*batteryLife*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.14.4 setParameter<service>

Description

The *setParameter<service>* command is used to request new device-originated bandwidth services and modify existing device-initiated services (now called "Timetables" in WirelessHART 7.4). Calling this command updates the mote's internal service table, which later initiates a request to the network manager for bandwidth allocation. A subsequent *serviceIndication* notification will be sent indicating the response from the network manager. The *getParameter<service>* command may be used to read the service table, including the state of the service request.

The *setParameter<service>* command may be sent at any time. If the network manager rejects a service request, the microprocessor can try again by re-issuing the *setParameter<service>* command.

To delete a service, set the time field of the desired service to zero. Service request flags, application domain, and destination address are ignored by the mote when time equals zero.

Normally all service requests are compared against the power limits set with the *setNVParameter<powerInfo>* command. Services that would cause the device to exceed its power budget are denied. In Manager 4.1.1, a service request of 1 ms will result in the manager respecting the power limit for publish services, but will allow a block-transfer service requests (see the SmartMesh WirelessHART User's Guide section on Services) that would result in a fast pipe being activated.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*service*)
serviceId	INT8U	none	Service identifier. 0x00-0x7F
serviceReqFlags	INT8U	Service Flags	Service request flag
appDomain	INT8U	Application Domain	Service application domain
destAddr	INT16U	none	Destination address. 0xF981 = gateway
time	INT32U	none	Period (msec), or latency (msec) if intermittent flag is set. Setting this value to 0 deletes the service.

Response

Parameter	Type	Enum	Description

rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*service*)
numServices	INT8U	none	Number of remaining service entries on mote

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.14.5 setParameter<hartDeviceStatus>

Description

The *setParameter<hartDeviceStatus>* command sets the current status of a WirelessHART device. The value passed in this parameter is used in all subsequent WirelessHART communications between the mote and the manager. This command is only required for WirelessHART-compliant devices. Refer to the HART Command Specifications for the appropriate value to use.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*hartDeviceStatus*)
hartDevStatus	INT8U	none	HART device status
hartExtDevStatus	INT8U	none	HART extended device status

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter identifier (*hartDeviceStatus*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.14.6 setParameter<hartDeviceInfo>

Description

The *setParameter<hartDeviceInfo>* command is used to set HART device information that the mote passes to gateway during join. This command must be issued prior to join. This command is only required for WirelessHART-compliant devices. Note that the contents of this command are not validated by mote.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*hartDeviceInfo*)
hartCmd0	INT8U[22]	none	HART Cmd 0
hartCmd20	INT8U[32]	none	HART Cmd 20

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Result code
paramId	INT8U	Parameter IDs	Parameter identifier (*hartDeviceInfo*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value

5.14.7 setParameter<eventMask>

Description

The *setParameter<eventMask>* command allows the microprocessor to subscribe to the types of events that may be sent in the mote's events notification message. This command may be called at any time and takes effect at the next event notification. The mote includes an event in the notification message if the corresponding bit in *<eventMask>* is set to "1," and excludes the event if the bit is set to "0." At mote reset, the default value of *<eventMask>* is "1" for all events.

New event type 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	Parameter IDs	Parameter identifier (*eventMask*)
eventMask	INT32U	Mote Events	Event mask

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Result code
paramId	INT8U	Parameter IDs	Parameter identifier (*eventMask*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_INVALID_LEN	Invalid packet length

5.14.8 setParameter<writeProtect>

Description

The *setParameter<writeProtect>* command allows the microprocessor to enable or disable access to selected WirelessHART commands via wireless or the *hartPayload* command. Refer to the SmartMesh WirelessHART User's Guide for the list of affected commands. If *writeProtect* is enabled and the mote receives any of these commands (either via wireless connection or via the *hartPayload* command), the command will have no effect, and the mote will return RC_7 (In Write Protect Mode). At mote boot, *writeProtect* is set to 0 (writes allowed). The current status of *writeProtect* may be read via the *getParameter<moteStatus>* command. This command is for WirelessHART-compliant devices only.

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter identifier (*writeProtect*)
writeProtect	INT8U	Write Protect Mode	Write protect mode

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Result code
paramId	INT8U	Parameter IDs	Parameter identifier (*writeProtect*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_INVALID_LEN	Invalid packet length

5.14.9 setParameter<lock>

Description

The *setParameter<lock>* command locks/unlocks select HART commands (ones that affect the configuration changed flag) to a specific master (GW or serial maintenance port) to prevent the other master from changing it. This command is intended for use when the lock is temporary, i.e. it does not persist through power cycle or reset. For nonvolatile locking, use the setNVParameter<lock> command. Note: This parameter is available in devices running mote software >= 1.1.0

Request

Parameter	Type	Enum	Description
paramId	INT8U	Parameter IDs	Parameter Identifier (*lock*)
code	INT8U	none	One of 0=unlock, 1=lock_master, 2=lock_all
master	INT16U	none	Short address of locking master (F981 for GW, 0000 for serial)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameter IDs	Parameter Identifier (*lock*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_UNKNOWN_PARAM	Unknown parameter value
RC_INVALID_VALUE	Invalid value

5.15 setNVParameter (0x03)

The *setNVParameter* command modifies persistent (non-volatile) parameters on the mote. Generally speaking, the settings changed with this command take effect after mote reset. Note: The flags field in the command header contains a "RAM" option that may be used with some commands to update the settings immediately.

5.15.1 setNVParameter<autojoin>

Description

The *setNVParameter<autojoin>* 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. (Available Mote >= 1.1)

> ⚠ 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
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NV parameter identifier (*autojoin*)
autojoin	INT8U	none	0 = off, 1 = on

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NV parameter identifier (*autojoin*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.2 setNVParameter<euCompliantMode>

Description

The *setNVParameter<euCompliantMode>* command may be used to enforce EN 300 328 duty cycle limits based on output power. This may cause the mote to 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 version >=1.2.x

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NV parameter identifier (euCompliantMode)
euCompliantMode	INT8U	none	Acceptable value are 0 = off (no duty cycling) and 1 = on (EN 300 328 compliant). 0 is the default.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NV parameter identifier (euCompliantMode)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.3 setNVParameter<hartAntennaGain>

Description

The *setNVParameter<hartAntennaGain>* command stores value of the antenna gain in the mote's persistent storage. This value is added to the conducted output power of the mote when replying to HART command 797 (Write Radio Power Output) and to HART command 798 (Read Radio Output Power). The antenna gain should take into account both the gain of the antenna and any loss (for example, attenuation from a long coax cable) between the mote and the antenna. By default, this value is 2, assuming a +2 dBi antenna gain. To change the transmit power immediately, use the write RAM option of this command.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*hartAntennaGain*)
antennaGain	INT8S	none	Antenna gain in dBi

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*hartAntennaGain*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.4 setNVParameter<hartCompliantMode>

Description

The *setNVParameter<hartCompliantMode>* command may be used to force strict compliance to HART specification requirements, specifically:

- join timeouts (faster in non-compliant mode)
- Keepalive interval (adapts to synch quality in non-compliant mode)
- Health report format (uses saturating counters in non-compliant mode)

Note: This parameter is referred to as *compliantMode* in documentation for versions 1.1.x, and *hartCompliantMode* in >=1.2.x, but the parameter ID is the same in both versions. The corresponding CLI command is called *mset compliantMode* in 1.1.x, and *mset hartCompliantMode* in >=1.2.x

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NV parameter identifier (hartCompliantMode)
hartCompliantMode	INT8U	none	Acceptable value are 0 = off (include SmartMesh improvements) and 1 = on (compliant). 0 is the default.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NV parameter identifier (hartCompliantMode)

Response Codes

Code	Description
RC_OK	Operation was successfully completed

RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.5 setNVParameter<joinKey>

Description

The *setNVParameter<joinKey>* command may be used to set the join key. Upon receiving this request, the mote stores the new join key in its persistent storage. Using the write RAM option will only have an effect if the command is called while the mote is in **Idle** state. Otherwise, the new value will be used after the next mote boot.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*joinKey*)
joinKey	SEC_KEY	none	New join key

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*joinKey*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.6 setNVParameter<joinShedTime>

Description

The *setNVParameter<joinShedTime>* command sets the join shed time used with HART command 771/772 to determine when the mote should transition between active and passive search. This command may be issued at any time and takes effect at the next mote boot. To change the join shed time immediately, use the write RAM option of this command, which can also be used at any time.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*joinShedTime*)
joinShedTime	INT32U	HART_TIME	Active search shed time. See HART spec 155 command 771/772

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*joinShedTime*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Read did not complete
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.7 setNVParameter<lock>

Description

The *setNVParameter<lock>* command persistently locks/unlocks select HART commands (ones that affect the configuration changed flag) to a specific master (GW or serial maintenance port) to prevent the other master from changing it. This command is intended for use when the lock persists through power cycle or reset. For temporary locking, use the *setParameter<lock>* command. Bit 7 in the flags field of the API header (see Packet Format) should be set (store in NV & RAM) when calling this command. Note: This parameter is available in devices running mote software >= 1.1.0

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NV parameter identifier (*lock*)
code	INT8U	none	One of 0=unlock, 1=lock_master, 2=lock_all
master	INT16U	none	Short address of locking master (F981 for GW, 0000 for serial)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NV parameter identifier (*lock*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.8 setNVParameter<macAddress>

Description

The *setNVParameter<macAddress>* command may be used to supersede the factory-configured MAC address of the mote.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NV parameter identifier (*macAddress*)
macAddr	MAC_ADDR	none	MAC address to use

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NV parameter identifier (*macAddress*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.9 setNVParameter<networkId>

Description

The *setNVParameter<networkId>* command may be used to set the persistent Network ID of the mote. The *networkId* is used to separate networks, and can be set during manufacturing or in the field. The mote reads this value from persistent storage at boot time. Note: while the mote is in **Idle** state, it is possible to update the value of mote's in-RAM Network ID by using the RAM flag in the header of this command. This avoids the extra reset that is needed to start using the Network ID. Network ID can also be set over the air using HART command 773 in a WirelessHART-compliant network.

As of version 1.1.1, 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 HART network ID - do not set the Manager's network ID to 0xFFFF.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identification (*networkId*)
networkId	INT16U	none	Network identifier

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identification (*networkId*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete

RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

5.15.10 setNVParameter<OTAPlockout>

Description

The *setNVParameter<OTAPlockout>* command specifies whether the mote's firmware can be updated over the air. Over-The-Air-Programming (OTAP) is allowed by default. The mote reads the *OTAPlockout* value from persistent storage at boot time. To change the value used currently, this command may be issued with RAM option.

Dust Networks recommends that OEMs allow their devices to receive firmware updates, either by leaving the *OTAPlockout* parameter at its default value, or by making *OTAPlockout* settable using a WirelessHART command that is available both over the air and through its wired maintenance port. OEMs have the option of making such a command password protected.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*OTAPlockout*)
otapLockout	INT8U	Otap Lockout	OTAP lockout setting

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*OTAPlockout*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed.
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value

RC_INVALID_VALUE	Parameter value failed validation

5.15.11 setNVParameter<powerInfo>

Description

The *setNVParameter<powerInfo>* command specifies the average current that is available to the mote. Using the write RAM option will only have an effect if the command is called while the mote is in **Idle** state. Otherwise, the new value will be used after the next mote boot.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParameterId	INT8U	Parameter IDs	NVParameter identifier (*powerInfo*)
powerSource	INT8U	Power Source	Power source
dischargeCur	INT16U	none	Discharge current (µA). The discharge current field indicates the maximum average current available to the mote. The manager will assign links such that this maximum average will not be exceeded, but may assign much lower as required by topology and data rates. This may limit the function of the device. For example, high-speed pipes may not be available if their use would exceed the discharge current.
dischargeTime	INT32U	none	Discharge time (sec). The discharge time field specifies the duration that the power source can supply the discharge current. In the case where the device is line-powered, or has a battery that will last longer than 37 hours, the discharge time will not be used in link assignment and should be set to 0xFFFFFFFF. Note that HART Spec-155 suggests setting to 24 hours if inapplicable.
recoverTime	INT32U	none	Recover time (sec). The recover time field specifies the time the device power supply takes to recharge under no load before the discharge current is available. If the power source is not rechargeable, recover time should be set to zero.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

nvParamId	INT8U	Parameter IDs	NVParameter identifier (*powerInfo*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_INVALID_VALUE	Parameter value failed validation

5.15.12 setNVParameter<ttl>

Description

The *setNVParameter<ttl>* command sets the mote's persistent packet Time To Live (TTL) value. TTL specifies the maximum number of hops a packet may traverse before it is discarded from the network. A mote sets the initial value of the TTL field in the packets it generates to this value. The mote reads the value from persistent storage at boot time. To change the TTL used currently, this command may be issued with the RAM option.

> ⚠ The mote defaults TTL to 127. For compliant devices, the HART specification currently defaults to 32, but this will change to 249 in spec version 7.4, as will the mote default. We suggest not changing the mote default unless HART specifically raises it as a compliance issue when you submit your device for testing.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*ttl*)
timeToLive	INT8U	none	Time To Live value

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*ttl*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete

RC_LOW_VOLTAGE	Voltage check failed
RC_UNKNOWN_PARAM	Unknown parameter value
RC_INVALID_VALUE	Parameter value failed validation

5.15.13 setNVParameter<txPower>

Description

The *setNVParameter<txPower>* command sets the mote output power. Refer to product datasheets for supported RF output power values. For example, if the mote has a typical RF output power of +8 dBm when the Power Amplifier (PA) is enabled, then set the *txPower* parameter to 8 to enable the PA. Similarly, if the mote has a typical RF output power of -2 dBm when the PA is disabled, then set the *txPower* parameter to -2 to turn off the PA. This command may be issued at any time and takes effect at the next mote boot. To change the transmit power immediately, use the write RAM option of this command, which can also be used at any time.

Request

Parameter	Type	Enum	Description
reserved	INT32U	none	Reserved field. Value = 0x00000000
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*txPower*)
txPower	INT8S	none	Transmit power, in dBm

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
nvParamId	INT8U	Parameter IDs	NVParameter identifier (*txPower*)

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_LEN	Invalid packet length
RC_WRITE_FAIL	Write did not complete
RC_LOW_VOLTAGE	Read did not complete
RC_UNKNOWN_PARAM	Unknown parameter value

5.16 testRadioTx (0x0B)

Description

The *testRadioTx* command initiates transmission over the radio. This command may only be issued prior to the mote joining the network. While executing this command the mote sends *numPackets* packets. Each packet consists of a payload of up to 125 bytes, and a 2-byte 802.15.4 CRC at the end. Bytes 0 and 1 contain the packet number (in big-endian format) that increments with every packet transmitted. Bytes 2..N contain a counter (from 0..N-2) that increments with every byte inside payload. Transmissions occur on the specified channel.

If number of packets parameter is set to 0x00, the mote will generate an unmodulated test tone on the selected channel. The test tone can only be stopped by resetting the mote.

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

⚠ Note: this command is deprecated and should not be used in new designs. The replacement command is *testRadioTxExt*.

Request

Parameter	Type	Enum	Description
channel	INT8U	none	RF channel number (0-15)
numPackets	INT16U	none	Number of packets to send(>=1). 0x00=send unmodulated test tone

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_VALUE	A parameter value is not valid

RC_INVALID_STATE	The mote is not in **Idle** state
RC_BUSY	Another test operation is in progress
RC_INVALID_LEN	Invalid packet size

5.17 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 mote must be reset (either hardware or software reset) after radio tests are complete and prior to joining.

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

⚠ Note: this command is deprecated and should not be used in new designs. The replacement command is *testRadioRxExt*.

Request

Parameter	Type	Enum	Description
channel	INT8U	none	RF channel number (0-15)
time	INT16U	none	Test duration (in seconds)

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_VALUE	A parameter value is not valid
RC_INVALID_STATE	The mote is not in **Idle** state
RC_BUSY	Another test operation is in progress

5.18 testRadioTxExt (0x13)

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	Test Type	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 (0-255) identifier for this radiotest transmitter.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_VALUE	A parameter value is not valid
RC_INVALID_STATE	The mote is not in **Idle** state
RC_BUSY	Another test operation is in progress
RC_INVALID_LEN	Invalid packet size

5.19 testRadioRxExt (0x14)

Description

The *testRadioRxExt* 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 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 0.

Response

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_INVALID_VALUE	A parameter value is not valid
RC_INVALID_STATE	The mote is not in **Idle** state

RC_BUSY	Another test operation is in progress
RC_INVALID_LEN	Invalid packet size

5.20 zeroize (0x15)

Description

The *zeroize* (zeroise) command erases flash area that is used to store configuration parameters, such as join keys. This command is intended to satisfy the zeroization requirement of the FIPS-140 standard. After the command executes, the mote should be reset. Available in mote >= 1.1.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

This section describes mote Notification packets. Notifications are sent by the mote when a significant change or event occurs on the device.

6.1 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-WHM 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-WHM 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.

Request

Parameter	Type	Enum	Description
utcTime	UTC_TIME	none	UTC time, seconds and microseconds
asn	ASN	none	The Absolute Slot Number, the number of timeslots since the manager booted
asnOffset	INT16U	none	The ASN offset is the number of microseconds since the beginning of the current slot

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_NO_RESOURCES	No resources are available to complete this command

6.2 serviceIndication

Description

The *serviceIndication* notification describes new manager-originated services (ID = 0x80-FF), or changes in existing services (ID = 0x00-7F). For more info on when the *serviceIndication* notification is sent and details about the individual parameters, see Bandwidth Services. If the time field contains the value *0x07FFFFFF*, the manager is unable to sustain the service due to network conditions and has effectively disabled the service. The service is not removed, however, and the microprocessor can elect to either delete the service or submit a request to update the service at a future time.

Request

Parameter	Type	Enum	Description
eventCode	INT8U	Event Code	Event code
netMgrRC	INT8U		HART response code from the manager
serviceId	INT8U	none	Service identifier. Value = 0x00-0xFF
serviceState	INT8U	Service State	Service state
serviceFlags	INT8U	Service Flags	Service flags
appDomain	INT8U	Application Domain	Application domain
destAddr	INT16U	none	Destination address
time	INT32U	none	Time

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_NO_RESOURCES	No resources are available to complete this command

6.3 events

Description

The *events* notification sends an event notification packet to the microprocessor informing it of new events that have occurred. The reported event is cleared from the mote when the mote receives an acknowledgement in the form of a response packet from the microprocessor.

Request

Parameter	Type	Enum	Description
events	INT32U	Mote Events	New events
state	INT8U	Mote State	Mote state
moteAlarms	INT32U	Mote Alarms	Current alarms

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_NO_RESOURCES	No resources are available to complete this command

6.4 dataReceived

Description

The *dataReceived* notification notifies the microprocessor that a packet was received. When the microprocessor receives a reliable *dataReceived* request, in addition to acknowledging the request with a *dataReceived* response it must also respond using the *send* command.

Request

Parameter	Type	Enum	Description
srcAddr	INT16U	none	Address of packet source. If this is a response to an end-to-end reliable request (as indicated in the flags byte), the source address must be used in the destination address field in the response sent via the send command.
seqNum	INT8U	none	Sequence number. The sequence number is the transport layer sequence number. If the data is an end-to-end reliable request (as indicated in the flags byte), the sequence number must be used in the response sent via the send command. If the data is not reliable, the sequence number will be zero.
pktLength	INT8U	none	Packet length
data	INT8U[]	none	HART data payload. See note below regarding contents.

Payload of the data field is formatted per Wireless HART specification, as follows:

Byte 0	Byte 1	Bytes 2-N
HART Status	HART Extended Status	<Cmd|Len|Payload><Cmd|Len|Payload><..>

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description

RC_OK	Operation was successfully completed
RC_NO_RESOURCES	No resources are available to complete this command

6.5 advReceived

Description

The *advReceived* notification notifies the microprocessor each time the mote receives an advertisement packet while in promiscuous listen mode. The command contains information about the advertisement, including the Network ID, Mote ID, RSSI, and join priority (hop depth). Note that *advReceived* notifications are sent only if the mote has been placed in listen mode using the search command (see search).

Request

Parameter	Type	Enum	Description
networkId	INT16U	none	Network ID
moteId	INT16U	none	Mote ID - a short address assigned to the mote by the manager at network join time
rssi	INT8S	none	RSSI
joinPri	INT8U	none	join priority (hop depth)

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description
RC_OK	Operation was successfully completed
RC_NO_RESOURCES	No resources are available to complete this command

6.6 suspendStarted

Description

The mote generates a *suspendStarted* notification when it enters the **Suspended** state as a result of processing Wireless HART command 972. When the suspend interval begins, the mote discontinues its radio operation and generates this notification. After the interval specified in command 972 ends, mote proceeds to reset. It is the responsibility of the attached microprocessor to re-join the network.

Request

Parameter	Type	Enum	Description
duration	INT32U	none	Duration of suspend interval (in seconds)

Response

Parameter	Type	Enum	Description
rc	INT8U	none	Response Code

Response Codes

Code	Description
RC_OK	Notification was processed
RC_NO_RESOURCES	Notification could not be processed due to lack of resources

7 Definitions

7.1 Command Identifiers

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. The Destination column indicates whether the packet is sent (or received) through the network or processed locally by the mote. Refer to the product datasheet for details on serial handshake signals.

Name	Value	Destination	Description
setParameter	0x01	Local	Sets parameters on the mote
getParameter	0x02	Local	Gets mote and network information
setNVParameter	0x03	Local	Stores a given parameter in the mote's persistent storage
getNVParameter	0x04	Local	Retrieves a given parameter from the mote's persistent storage
send	0x05	Network	Packet destined for the network
join	0x06	Local	Requests that mote attempt to join the network
disconnect	0x07	Local	Requests that mote disconnect from the network
reset	0x08	Local	Resets mote
lowPowerSleep	0x09	Local	Shuts down peripherals and puts mote into deep sleep mode
hartPayload	0x0A	Local	Allows the microprocessor to forward a HART payload to the mote
testRadioTx	0x0B	Local	Allows the serial device to initiate a series of packet transmissions (deprecated by *testRadioTxExt*)
testRadioRx	0x0C	Local	Allows the serial device to test radio reception for a specified channel (deprecated by *testRadioRxExt*)
clearNV	0x10	Local	Resets the mote non-volatile memory (NV) to its factory default state
search	0x11	Local	Causes the mote to listen for advertisements and send a notification for each it receives
testRadioTxExt	0x13	Local	Allows the serial device to initiate a number of transmission tests
testRadioRxExt	0x14	Local	Allows the serial device to test radio reception for a specified channel

zeroize	0x15	Local	Erases flash area that contains configuration parameters, including join keys
fileWrite	0x17	Local	Write to the scratchpad file
fileRead	0x18	Local	Write to the scratchpad file
fileOpen	0x19	Local	Open the scratchpad file in the mote file system

7.2 Notification Identifiers

Name	Value	Destination	Description
timeIndication	0x0D	Local	Time and mote state information.
serviceIndication	0x0E	Local	Notifies the microprocessor of changes in service status.
events	0x0F	Local	Notifies the microprocessor that a new event has occurred.
advReceived	0x12	Local	Notifies the microprocessor that an advertisement has been received.
suspended	0x16	Local	Indicates that the mote entered suspended state as a result of receiving command 972 (suspend)
dataReceived	0x81	Network	Mote forwards a packet from the network to the microprocessor.

⚠ 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.

7.3 Parameter Identifiers

Name	Value	Description	Get	Set	NV
macAddress	0x01	MAC address in the non-volatile memory of the mote	x	x	x
joinKey	0x02	Network join key in the non-volatile memory of the mote. (Not readable for security reasons.)		x	x
networkId	0x03	Mote's Network ID in the non-volatile memory of the mote	x	x	x
txPower	0x04	RF output power for the device	x	x	x
powerInfo	0x05	Power supply for the device	x	x	x
joinDutyCycle	0x06	Allows the serial processor to control the ratio of active listen time to doze time (a low-power radio state) during network search		x	
batteryLife	0x07	Allows the serial processor to update the remaining battery life information that the mote reports to the manager		x	
service	0x08	Allows the device to initiate a service request or update an existing service.	x	x	
hartDeviceStatus	0x09	Current status of the Wireless HART device		x	
hartDeviceInfo	0x0A	Wireless HART device information to be used by the mote when it joins the network		x	
eventMask	0x0B	Allows the serial processor to disable events that may be sent in the mote's event notification message		x	
moteInfo	0x0C	Static information about the mote's hardware and software	x		
networkInfo	0x0D	Mote's current network-related parameters	x		
moteStatus	0x0E	Mote's state and frequently changing information	x		
time	0x0F	Current time on the mote	x		
charge	0x10	Mote's charge consumption	x		
testRadioRxStats	0x11	Results of the mote radio reception test	x		
writeProtect	0x12	Enables/disables write protection of non-volatile configuration parameters (readable via *getParameter<moteStatus>*)		x	
ttl	0x13	Time To Live that the mote uses for all originating packets. The default is 127 if this is not set	x	x	x

hartAntennaGain	0x14	Gain of the antenna attached to the mote	x	x	x
OTAPlockout	0x15	Sets whether a mote can be Over-The-Air-Programmed (OTAP)	x	x	x
reserved	0x16	This parameter id is reserved and should not be used			
hrCounterMode	0x17	Controls whether the mote should use saturating or rollover counters for health reports	x	x	x
autojoin	0x18	Causes a mote in slave mode to join by itself after boot.	x	x	x
hartCompliantMode	0x19	Controls whether the mote behavior should strictly comply with HART standards (was called "compliantMode" in previous documentation)	x	x	x
lock	0x1A	Locks commands that affect CCF to a particular master (supports HART commands 71 and 76)	x	x	x
euCompliantMode	0x1B	Controls whether the mote should duty cycle to comply with EN 300 328 limits based on the output power	x	x	x
joinShedTime	0x1C	Used with HART command 771/772 to determine when the mote should transition between active and passive search	x	x	x

7.4 Response Codes

An unrecognized response code (e.g. a new code not listed here) should be treated as a general error.

Value	Name	Description
0	RC_OK	Operation was successfully completed.
3	RC_BUSY	Resource unavailable
4	RC_INVALID_LEN	Invalid length
5	RC_INVALID_STATE	Invalid state
6	RC_UNSUPPORTED	Command or operation not supported
7	RC_UNKNOWN_PARAM	Unknown parameter
8	RC_UNKNOWN_CMD	Unknown command
9	RC_WRITE_FAIL	Write did not complete
10	RC_READ_FAIL	Read did not complete
11	RC_LOW_VOLTAGE	Voltage check failed
12	RC_NO_RESOURCES	No resources are available to complete command
13	RC_INCOMPLETE_JOIN_INFO	Incomplete join info
14	RC_NOT_FOUND	Resource not found
15	RC_INVALID_VALUE	Invalid value
16	RC_ACCESS_DENIED	Access to this command/variable denied
18	RC_OPEN_FAIL	Open operation failed
19	RC_ERASE_FAIL	Erase operation failed

7.5 Service Flags

BitMask	Name	Description
0x01	Source	Mote is source of data generated
0x02	Sink	Mote is receiver of data
0x04	Intermittent	Intermittent traffic (as opposed to regular reporting)

The following table shows supported combinations of the service request flags by application domain.

Application Domain	Source	Sink	Intermittent
Publish	1	0	0
Event	1	0	1
Block Transfer	0 or 1	0 or 1	0
Maintenance*			
* Any value for maintenance application domain will be denied.			

7.6 Application Domain

Value	Name
0x00	Publish
0x01	Event
0x02	Maintenance
0x03	Block transfer

7.7 Power Status

Value	Name
0x00	Nominal
0x01	Low
0x02	Critically low
0x03	Recharging low
0x04	Recharging high

7.8 Mote Events

BitMask	Event	Description
0x01	Boot	The mote booted up
0x02	Alarms changed	Value of alarms field changed
0x04	Time changed	UTC reference on the mote changed (see "Timestamps" in the SmartMesh WirelessHART User's Guide)
0x08	Join failed	Join operation failed
0x10	Disconnected	The mote has disconnected from the network
0x20	Operational	The mote has a connection to the gateway to send data
0x40	Configuration changed	A write command changed the mote configuration
0x100	Join Started	The mote has sent a join request

7.9 Write Protect Mode

Value	Name
0x00	Write allowed
0x01	Write not allowed

7.10 Mote State

Value	Name	Description
0x00	Init	The mote is in the process of booting
0x01	Idle	The mote is accepting configuration commands. Upon receiving a join command, the mote moves into the **Searching** state. The **Idle** state is equivalent to the HART "Low Power" state.
0x02	Searching	The mote's receiver is on with a configurable duty cycle while the mote is actively searching for a network. The **Searching** state is equivalent to the HART "Active Search" state or "Passive Search," depending on the duty cycle setting.
0x03	Negotiating	The mote has detected a network and has sent a join request to the manager
0x04	Connected	The mote has joined the network and established communication with the network manager. The **Connected** state is equivalent to the HART "Quarantine" state.
0x05	Operational	The mote has links to both the network manager and gateway, and is ready to send data
0x06	Disconnected	The mote has disconnected from the network
0x07	Radio test	The mote is in radio test mode
0x08	Promiscuous Listen	The mote received a *search* command and is in promiscuous listen mode
0x09	Suspended	The mote entered suspended state after having processed command 972

7.11 Mote Alarms

BitMask	Alarm	Description
0x01	NV Error	Error related to flash access
0x02	Low voltage	Mote supply voltage is below operating minimum
0x04	OTP Error	Detected an error in factory-programmed settings
0x08	Not ready to send	Mote can't forward data packets to the network (usually due to congestion) - this alarm is not generated in serial modes 1 or 3

7.12 Status Flags

Bit	Description
7-1	Reserved (use 0's for these bits)
0	Write Protect (see *setParameter<writeProtect>*): 0 = Write allowed 1 = Write not allowed

7.13 Service State

Service pending bit (bit 7):

Value	Name
0	No service pending
1	Service request pending

Service state (bits 6-0):

Value	Description
0x00	Inactive
0x01	Active
0x02	Requested

7.14 setNVParameter Request Flag

Bit	Description
7	Write to RAM: 0 = Write NV only 1 = Write NV and RAM
6-3	Reserved (use 0's for these bits)
2-0	See Flags in Packet Field Descriptions for bit description

7.15 Power Source

The power source field describes whether the device is line, battery, or rechargeable/scavenging powered. A line-powered device has a hardwired connection to a plant power source. A battery-powered device operates solely off its (internal) battery. A device powered by rechargeable battery power or by energy scavenging has a short-term supply of energy (at least one hour) that is continuously being replenished. The device's power is being replenished by harvesting and converting energy from the environment surrounding the device (for example, solar, vibration, or heat).

Value	Description
0x00	Line
0x01	Battery
0x02	Rechargeable/Scavenging

7.16 OTAP Lockout

Value	Description
0x00	OTAP allowed (default)
0x01	OTAP disabled

7.17 Priority

Value	Description
0x00	Low
0x01	Medium
0x02	High

7.18 Send Request and dataReceived notification flags

Bit	Description
7	Transport direction: 0 = End-to-end request 1 = End-to-end response
6	Transport type: 0 = Best effort 1 = Reliable *
5-3	Reserved (use 0's for these bits)
2-0	See Flags in Packet Field Description for bit description
*Only used for end-to-end response to a reliable *dataReceived* command request (see See Reliable Communications from Controller to Microprocessor). Microprocessor-initiated end-to-end reliable requests are not supported.	

7.19 Test Type

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

7.20 Event Code

Value	Description
0	Created/updated
1	Deleted
2	Rejected
3	Delayed response

7.21 HR Counter Mode

Value	Description
0	Rollover
1	Saturating

7.22 HART Compliance Mode

Value	Description
0	Some timers and counters deviate from HART specification
1	Strict HART compliance

7.23 File Open Options

Value	Description
0x01	Create the file if it does not exist

7.24 File Open Mode

Value	Description
0x01	Others have read permission
0x02	Others have write permission
0x03	Others have read/write permission
0x04	This is a temporary file (it is deleted upon reset or power cycle)
0x08	This file is created in shadow mode (for wear leveling)

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.