



SmartMesh IP Mote API Guide

Advance Information

This document contains advance information of a product in development. All specifications are subject to change without notice. Consult LTC factory before using.



Table of Contents


1	About This Guide	6
1.1	Related Documents	6
1.2	Conventions used	6
1.3	Related Documents	7
1.4	Conventions Used	7
1.5	Revision History	8
2	Protocol	9
2.1	Data Representation	9
2.1.1	Common Data Types	9
2.1.2	Integer representation	10
2.1.3	Transmission Order	11
2.2	Packet Format	11
2.2.1	HDLC Packet Encapsulation	11
2.2.2	HDLC Payload Contents	13
2.2.3	Concatenated commands	14
2.3	Communication between Mote and Microprocessor	15
2.3.1	Acknowledged link	15
2.3.2	Guidelines for forward-compatible clients	15
3	Commands	17
3.1	getParam	17
3.1.1	getParam<macAddress>	18
3.1.2	getParam<networkId>	18
3.1.3	getParam<txPower>	19
3.1.4	getParam<joinDutyCycle>	
3.1.5	getParam<eventMask>	21
3.1.6	getParam<moteInfo>	21
3.1.7	getParam<netInfo>	22
3.1.8	getParam<moteStatus>	23
3.1.9	getParam<time>	24
3.1.10	getParam<charge>	25
3.1.11	getParam<testRadioRxStats>	26
3.1.12	getParam<OTAPLockout>	27
3.1.13	getParam<shortAddress>	28
3.1.14	getParam<ipv6Address>	28
3.1.15	getParam<routingMode>	29
3.1.16	getParam<pwrSrcInfo>	30
3.1.17	getParam<autoJoin>	31
3.2	setParam	32
3.2.1	setParam<macAddress>	32



3.2.2	setParam<joinKey>	33
3.2.3	setParam<networkId>	34
3.2.4	setParam<txPower>	34
3.2.5	setParam<joinDutyCycle>	35
3.2.6	setParam<eventMask>	36
3.2.7	setParam<OTAPLockout>	37
3.2.8	setParam<routingMode>	38
3.2.9	setParam<pwrSrcInfo>	38
3.2.10	setParam<autoJoin>	39
3.3	join	40
3.4	disconnect	41
3.5	reset	42
3.6	lowPowerSleep	42
3.7	testRadioTx	43
3.8	testRadioRx	44
3.9	clearNv	45
3.10	requestService	
3.11	getServiceInfo	47
3.12	openSocket	48
3.13	closeSocket	48
3.14	bindSocket	49
3.15	sendTo	50
3.16	search	51
4	Notifications	52
4.1	advReceived	52
4.2	events	52
4.3	receive	53
4.4	timeIndication	53
4.5	txDone	54
5	Definitions	55
5.1	Commands	55
5.2	Parameters	55
5.3	Notifications	56
5.4	Response Codes	56
5.5	Service Types	57
5.6	Service States	57
5.7	Protocol Types	58
5.8	Packet Priorities	58
5.9	Mote States	58
5.10	Events	59
5.11	Alarms	59
5.12	Radio test types	59
5.13	Packet transmit status	60



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

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

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

Disclaimer

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

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

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

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

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



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

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

© Dust Networks, Inc. 2012 All Rights Reserved.

- [About this Guide \(IPMT_API\)](#)
- [Protocol \(IPMT_API\)](#)
- [Commands \(IPMT_API\)](#)
- [Notifications \(IPMT_API\)](#)
- [Definitions \(IPMT_API\)](#)



1 About This Guide

This guide describes the IP Mote Serial API.

1.1 Related Documents

The following documents are available for the SmartMesh-enabled network:

The following documents are available for the SmartMesh-enabled network:

- SmartMesh IP Quick Start Guide
- SmartMesh IP Network User Guide
- SmartMesh IP Manager User Guide
- SmartMesh IP Manager CLI Guide
- SmartMesh IP Manager API Guide
- SmartMesh IP Mote User Guide
- [SmartMesh IP Mote API Guide](#)
- SmartMesh IP Mote CLI Guide

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, and menu commands.

Italic type is used to introduce a new term



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 or API

1.3 Related Documents

The following documents are available for the SmartMesh-enabled network:

- SmartMesh IP Quick Start Guide
- SmartMesh IP Network User Guide
- SmartMesh IP Manager User Guide
- SmartMesh IP Manager CLI Guide
- SmartMesh IP Manager API Guide
- SmartMesh IP Mote User Guide
- [SmartMesh IP Mote API Guide](#)
- SmartMesh IP Mote CLI Guide

1.4 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, and menu commands.

Italic type is used to introduce a new term



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 or API

1.5 Revision History

Revision	Date	Description
----------	------	-------------



2 Protocol

- [Data Representation \(IPMT_API\)](#)
- [Packet Format \(IPMT_API\)](#)
- [Communication between Mote and Microprocessor](#)

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

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

2.1 Data Representation

- [Common Data Types](#)
- [Integer representation](#)
- [Transmission Order](#)

2.1.1 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.
INT8	1	Signed byte or character.
INT16	2	Short signed integer.
INT32	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: <ul style="list-style-type: none">• 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: <ul style="list-style-type: none">• 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.

2.1.2 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.

INT8, INT8U

Bit 7.. Bit 0

INT16, INT16U

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

INT32, 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
----------------	----------------	----------------	---------------	--------------

2.1.3 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).

2.2 Packet Format

- [HDLC Packet Encapsulation](#)
 - [HDLC Encoding example](#)
 - [HDLC Decoding example](#)
- [HDLC Payload Contents](#)
 - [API Header](#)
 - [Flags field](#)
 - [Request/Response flag](#)
 - [Packet id](#)
 - [Sync flag](#)
 - [API Payload](#)
 - [Maximum packet size](#)
- [Concatenated commands](#)

2.2.1 HDLC Packet Encapsulation

HDLC protocol is used for all API communication between mote and serial microprocessor. All packets are encapsulated in HDLC framing described in [RFC1662](#). Packets start and end with a 0x7E flag, and contain a 16-bit CRC-CCITT Frame Check Sequence (FCS).

Flag	Payload	FCS	Flag
7E	Packet Payload	2 bytes	7E

Note that packets do not contain HDLC Control and Address fields that are mentioned in [RFC1662](#).



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:

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.

Payload04 03 01 00 03 00 **7E**

2.2.2 HDLC Payload Contents

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

API Header**API Payload**

HDLC Payload Format

API Header

The API header contains the following fields:

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

Flags field

Flags is an INT8U field containing the following fields:

Bit**Description**



0 (LSB)	0=Request, 1=Response
1	Packet id
2	<i>Reserved, set to 0</i>
3	Sync
4	<i>Reserved, set to 0</i>
5	<i>Reserved, set to 0</i>
6	<i>Reserved, set to 0</i>
7 (MSB)	<i>Reserved, set to 0</i>

Request/Response flag

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

Packet id

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

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

Sync flag

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

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

API Payload

The API Payload portion of the packet contains request or response payloads listed in the [Commands](#) and [Notifications](#) sections.

Maximum packet size

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



2.2.3 Concatenated commands

The mote serial protocol supports concatenation of commands. This enables OEMs to optimize the number of API exchanges required to set or get information. Multiple request packets may be concatenated into a single HDLC message, as follows:

API Header #1	API Request Payload #1	API Header #2	API Request Payload #2	API Header #n	API Request Payload #n
---------------	------------------------	---------------	------------------------	------	---------------	------------------------

The response to such request would be formatted similarly:

API Header #1	API Response Payload #1	API Header #2	API Response Payload #2	API Header #n	API Response Payload #n
---------------	-------------------------	---------------	-------------------------	------	---------------	-------------------------



Requests and Responses may not be mixed in one packet.

The total length of request and response packets may not exceed maximum packet size or an error will be returned.

2.3 Communication between Mote and Microprocessor

- [Acknowledged link](#)
- [Guidelines for forward-compatible clients](#)

2.3.1 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 the *packet id* field if a new packet is sent, and leave the field unchanged for retransmissions.

The microprocessor should re-transmit packets if

- no response is received
- RC_NO_RESOURCES error code is received in response packet (indicating that the other side is not ready to process the packet)

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



2.3.2 Guidelines for forward-compatible clients

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

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

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

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

If the protocol changes in any other incompatible way, the protocol version reported via [getParam<moteInfo>](#) will be changed.



3 Commands

- [getParam \(IPMT_API\)](#)
- [setParam \(IPMT_API\)](#)
- [join \(IPMT_API\)](#)
- [disconnect \(IPMT_API\)](#)
- [reset \(IPMT_API\)](#)
- [lowPowerSleep \(IPMT_API\)](#)
- [testRadioTx \(IPMT_API\)](#)
- [testRadioRx \(IPMT_API\)](#)
- [clearNv \(IPMT_API\)](#)
- [requestService \(IPMT_API\)](#)
- [getServiceInfo \(IPMT_API\)](#)
- [openSocket \(IPMT_API\)](#)
- [closeSocket \(IPMT_API\)](#)
- [bindSocket \(IPMT_API\)](#)
- [sendTo \(IPMT_API\)](#)
- [search \(IPMT_API\)](#)

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

API Header	API Payload
------------	-------------

Command contents

3.1 getParam

- [getParam<macAddress> \(IPMT_API\)](#)
- [getParam<networkId> \(IPMT_API\)](#)
- [getParam<txPower> \(IPMT_API\)](#)
- [getParam<joinDutyCycle> \(IPMT_API\)](#)
- [getParam<eventMask> \(IPMT_API\)](#)
- [getParam<moteInfo> \(IPMT_API\)](#)
- [getParam<netInfo> \(IPMT_API\)](#)
- [getParam<moteStatus> \(IPMT_API\)](#)
- [getParam<time> \(IPMT_API\)](#)
- [getParam<charge> \(IPMT_API\)](#)
- [getParam<testRadioRxStats> \(IPMT_API\)](#)
- [getParam<OTAPLockout> \(IPMT_API\)](#)



- [getParam<shortAddress>](#) (IPMT_API)
- [getParam<ipv6Address>](#) (IPMT_API)
- [getParam<routingMode>](#) (IPMT_API)
- [getParam<pwrSrcInfo>](#) (IPMT_API)
- [getParam<autoJoin>](#) (IPMT_API)

3.1.1 getParam<macAddress>

Description

This command returns the MAC address of the device. By default, the MAC address returned is the EUI64 address of the device assigned by mote manufacturer, but it may be overwritten using the [setParam<macAddress>](#) command.

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (macAddress)

Response Payload

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

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.2 getParam<networkId>



Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (networkId)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response code
paramId	INT8U	Parameters	Parameter id (networkId)
networkId	INT16U	none	Network Identifier

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.3 getParam<txPower>

Description

Get the radio output power in dBm, excluding any antenna gain.

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (txPower)

**Response Payload**

Parameter	Type	Enum	Description
rc	INT8U	none	Response code
paramId	INT8U	Parameters	Parameter id (txPower)
txPower	INT8	none	Transmit power, in dBm

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_INVALID_VALUE	Invalid value of power specified

3.1.4 getParam<joinDutyCycle>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Param id (joinDutyCycle)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response code
paramId	INT8U	Parameters	Parameter id (joinDutyCycle)
joinDutyCycle	INT8U	none	Duty cycle (0-255), where 0=0.2% and 255=99.8%



Response Codes

Code	Description
RC_OK	Command completed successfully
RC_UNKNOWN_PARAM	Unknown paramId value

3.1.5 getParam<eventMask>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (eventMask)

Response Payload

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

Response Codes

Code	Description
RC_OK	Command completed successfully



3.1.6 getParam<moteInfo>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter Id (moteInfo)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter Id (moteInfo)
apiVersion	INT8U	none	Version of the api protocol
serialNumber	INT8U[8]	none	Serial number of the device
hwModel	INT8U	none	Hardware model
hwRev	INT8U	none	Hardware revision
swVerMajor	INT8U	none	Software version: major
swVerMinor	INT8U	none	Software version: minor
swVerPatch	INT8U	none	Software version: patch
swVerBuild	INT16U	none	Software version: build
bootSwVer	INT8U	none	Bootloader software version

Response Codes

Code	Description
RC_OK	Command completed successfully



3.1.7 getParam<netInfo>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (netInfo)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter id (netInfo)
macAddr	MAC_ADDR	none	MAC address of the device
shortAddr	INT16U	none	Short address of the device, same as moteld
networkId	INT16U	none	Network Identifier
slotSize	INT16U	none	Slot size (microseconds)

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.8 getParam<moteStatus>

Description

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



Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (moteStatus)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter id (moteStatus)
state	INT8U	Mote States	Mote state
reserved	INT8U[3]	none	This field should be ignored
numParents	INT8U	none	Number of parents
alarms	INT32U	Alarms	Current alarms (bitmap)
reserved	INT8U	none	This field should be ignored

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.9 getParam<time>

Description

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

Request Payload

--	--	--	--



Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (time)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter id (time)
upTime	INT32U	none	Time from last reset (seconds)
utcTime	UTC_TIME	none	UTC time
asn	ASN	none	Time since start of network (slots)
asnOffset	INT16U	none	Time from the start of the asn (microseconds)

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.10 getParam<charge>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (charge)

**Response Payload**

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code
paramId	INT8U	Parameters	Parameter id (charge)
qTotal	INT32U	none	Charge since last reset (millicoulombs)
upTime	INT32U	none	Time since reset (seconds)
tempInt	INT8	none	Temperature (integral part, in °C)
tempFrac	RestrParamType	none	Temperature (fractional part, in 1/255 of °C)

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.11 **getParam<testRadioRxStats>**

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (testRadioRxStats)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code



paramId	INT8U	Parameters	Parameter id (testRadioRxStats)
rxOk	INT16U	none	Number of packets successfully received
rxFailed	INT16U	none	Number of packets with error(s) received (CRC)

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.12 getParam<OTAPLockout>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (OTAPLockout)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response Code
paramId	INT8U	Parameters	Param id (OTAPLockout)
mode	BOOL	none	0=OTAP allowed, 1=OTAP not allowed

Response Codes

Code	Description
------	-------------



RC_OK	Command completed successfully
-------	--------------------------------

3.1.13 getParam<shortAddress>

Description

This command retrieves the short address of the mote. If the mote is not in the network, short address of 0 is returned.

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (shortAddress)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter id (shortAddress)
address	INT16U	none	Short address

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.14 getParam<ipv6Address>

Description

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



Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Param id (ipv6Address)

Response Payload

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

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.15 getParam<routingMode>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (routingMode)

Response Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------



rc	INT8U	Response codes	Response code
paramId	INT8U	Parameters	Parameter id (routingMode)
routingMode	BOOL	none	Routing mode (0=enable routing, 1=disable routing)

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.16 getParam<pwrSrcInfo>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (powerSrcInfo)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter id (pwrSrcInfo)
maxStCurrent	INT16U	none	Maximum steady-state current in uA; 0xFFFF=no limit
minLifetime	INT8U	none	Minimum lifetime, in months; 0=no limit
currentLimit_0	INT16U	none	Current limit, in uA; 0=limit entry empty
dischargePeriod_0	INT16U	none	Discharge period, in seconds
rechargePeriod_0	INT16U	none	Recharge period, in seconds



currentLimit_1	INT16U	none	Current limit, in uA; 0=limit entry empty
dischargePeriod_1	INT16U	none	Discharge period, in seconds
rechargePeriod_1	INT16U	none	Recharge period, in seconds
currentLimit_2	INT16U	none	Current limit, in uA; 0=limit entry empty
dischargePeriod_2	INT16U	none	Discharge period, in seconds
rechargePeriod_2	INT16U	none	Recharge period, in seconds

Response Codes

Code	Description
RC_OK	Command completed successfully

3.1.17 getParam<autoJoin>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (autoJoin)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter id (autoJoin)
autoJoin	BOOL	none	0=do not auto join; 1=auto join



Response Codes

Code	Description
RC_OK	The command completed successfully

3.2 setParam

- [setParam<macAddress> \(IPMT_API\)](#)
- [setParam<joinKey> \(IPMT_API\)](#)
- [setParam<networkId> \(IPMT_API\)](#)
- [setParam<txPower> \(IPMT_API\)](#)
- [setParam<joinDutyCycle> \(IPMT_API\)](#)
- [setParam<eventMask> \(IPMT_API\)](#)
- [setParam<OTAPLockout> \(IPMT_API\)](#)
- [setParam<routingMode> \(IPMT_API\)](#)
- [setParam<pwrSrcInfo> \(IPMT_API\)](#)
- [setParam<autoJoin> \(IPMT_API\)](#)

3.2.1 setParam<macAddress>

Description

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

Request Payload

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

Response Payload

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



Response Codes

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

3.2.2 setParam<joinKey>

Description

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

Request Payload

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

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
paramId	RestrParamType	Parameters	Parameter Id (joinKey)

Response Codes

Code	Description
------	-------------



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

3.2.3 setParam<networkId>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (networkId)
networkId	INT16U	none	Network Identifier

Response Payload

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

Response Codes

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

3.2.4 setParam<txPower>

Description

The setParameter<txPower> command sets the mote output power. This setting is persistent. The command

may be issued at any time and takes effect on next transmission. 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.

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (txPower)

Response Payload

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

Response Codes

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

3.2.5 setParam<joinDutyCycle>

Description

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

Request Payload



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

Response Payload

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

Response Codes

Code	Description
RC_OK	Command completed successfully

3.2.6 setParam<eventMask>

Description

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

Request Payload

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

Response Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------



rc	INT8U	Response Codes	Response code
paramId	INT8U	Parameters	Parameter id (eventMask)

Response Codes

Code	Description
RC_OK	Command completed successfully

3.2.7 setParam<OTAPLockout>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter id (OTAPLockout)
mode	BOOL	none	0=otap allowed, 1=OTAP not allowed

Response Payload

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

Response Codes

Code	Description
RC_OK	Command completed successfully



RC_INVALID_VAL	Invalid value of mode parameter
RC_WRITE_ERROR	Couldn't update persistent storage

3.2.8 setParam<routingMode>

Description

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

Request Payload

Parameter	Type	Enum	Description
paramId	INT8U	Parameters	Parameter Id (routingMode)
mode	BOOL	none	0=routing, 1=non-routing

Response Payload

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

Response Codes

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

3.2.9 setParam<pwrSrcInfo>

Description

This command allows the microprocessor to configure power source information on the device. This setting is



persistent and is used at network join time.

Request Payload

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

Response Payload

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

Response Codes

Code	Description
RC_OK	Command completed successfully



3.2.10 setParam<autoJoin>

Description

This command allows the microprocessor to change between automatic and manual joining of the mote. In manual mode, an explicit [join](#) command is required to initiate joining. This setting is persistent and takes effect after mote reset.

Request Payload

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

Response Payload

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

Response Codes

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

3.3 join

Description

The *join* command requests that mote start searching for the network and attempt to join. The mote must be in the IDLE state for this command to be valid. Note that the join time will be affected by the maximum current setting.



Request Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response Code

Response Codes

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

3.4 disconnect

Description

The *disconnect* command requests that the mote initiate disconnection from the network. After disconnection completes, the mote will generate a disconnected event, and proceed to reset. If the mote is not in the network, the disconnected event will be generated immediately. This command may be issued at any time.

Request Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------

Response Payload

--	--	--	--



Parameter	Type	Enum	Description
rc	INT8U	none	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted

3.5 reset

Description

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

Request Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted

3.6 lowPowerSleep



Description

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

Request Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response Code

Response Codes

Code	Description
RC_OK	Command was accepted

3.7 testRadioTx

Description

The *testRadioTx* command allows the microprocessor to initiate a radio transmission sequence that may be used for testing or regulatory certification. This command may only be issued in when the mote is in IDLE mode. The mote must be reset (either hardware or software reset) after each type of test and prior to joining.

Request Payload

Parameter	Type	Enum	Description
type	INT8U	Test Types	Type of test: Packet, Continuous Modulation (CM), Continuous Wave (CW)



chanMask	INT16U	none	Transmission channel(s). Bit 0=channel 0 (2405 MHz), Bit 1=channel 1(2410 MHz), etc. CM and CW mode may only be used with a single channel specified. Packet transmission will occur on multiple channels, one packet on each channel in round-robin sequence.
numPackets	INT16U	none	Number of packets to transmit. This parameter is only used for packet test.
packetLen	INT8U	none	Length of packet to transmit (2-125). This parameter is only used for packet test.
txPower	INT8	none	Transmit power, in dB. Valid values are 0 and 8.

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

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

3.8 testRadioRx

Description

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

Request Payload



Parameter	Type	Enum	Description
channel	INT8U	none	Channel to receive on (0-14).
time	INT16U	none	Test duration (in seconds).

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description
RC_OK	Command was accepted.
RC_INVALID_VALUE	The mote is not in IDLE state
RC_BUSY	Another test operation in progress

3.9 clearNv

Description

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

Request Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------

Response Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------



rc	INT8U	none	Return code
----	-------	------	-------------

Response Codes

Code	Description
RC_OK	Command completed successfully
RC_WRITE_FAIL	Flash operation failed

3.10 requestService

Description

The *requestService* command may be used to request a new or changed service level to a destination device in the mesh. This command may only be used to update service to a device with an existing connection (session). A service is uniquely identified by <destination,type> pair.

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

Request Payload

Parameter	Type	Enum	Description
destAddr	INT16U	none	Address of in-mesh destination of service. The manager address (0xFFFE) is the only value currently supported
serviceType	INT8U	Service Types	Service type
value	INT32U	none	Inter-packet interval (in ms)

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response codes	Response code



Response Codes

Code	Description
RC_OK	Service request accepted

3.11 getServiceInfo

Description

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

Request Payload

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

Response Payload

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

Response Codes

Code	Description
------	-------------



RC_OK	Command successfully completed
-------	--------------------------------

3.12 openSocket

Description

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

Request Payload

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

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	Response Codes	Response code
socketId	INT8U	none	Socket id

Response Codes

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

3.13 closeSocket

Description

Close the previously open socket.

Request Payload

--	--	--	--



Parameter	Type	Enum	Description
socketId	INT8U	none	socket id

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description
RC_OK	Socket successfully closed
RC_NOT_FOUND	Socket id not found

3.14 bindSocket

Description

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

Request Payload

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

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response code



Response Codes

Code	Description
RC_OK	Operation completed successfully
RC_NOT_FOUND	Invalid socket id

3.15 sendTo

Description

Send a packet into the network. If the command returns RC_OK, the mote has accepted the packet and has queued it up for transmission. A [txDone](#) notification will be issued when the packet has been sent, if and only if the packet ID passed in this command is different from 0xffff. You can set the packet ID to any value. The notification will contain the packet ID of the packet just sent, allowing association of the notification with a particular packet.

Request Payload

Parameter	Type	Enum	Description
socketId	INT8U	none	Socket id
destAddr	IPV6_ADDR	none	Destination IPV6 address
destPort	INT16U	none	Destination port
serviceType	INT8U	Service Types	Service type
priority	INT8U	Packet Priorities	Priority of the packet
packetId	INT16U	none	User-defined packet ID for txDone notification; 0xFFFF=do not generate notification
payload	INT8U[]	none	Payload of the packet

Response Payload

--	--	--	--



Parameter	Type	Enum	Description
rc	INT8U	none	Response code

Response Codes

Code	Description
RC_OK	Packet was queued up for transmission
RC_NO_RESOURCES	No queue space to accept the packet

3.16 search

Description

The *search* command requests that mote start listening for advertisements and report those heard from any network without attempting to join. The mote must be in the IDLE state for this command to be valid. The search state can be exiting by issuing the *join* command or the *reset* command.

Request Payload

Parameter	Type	Enum	Description
-----------	------	------	-------------

Response Payload

Parameter	Type	Enum	Description
rc	INT8U	none	Response Code

Response Codes

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



4 Notifications

- [advReceived \(IPMT_API\)](#)
- [events \(IPMT_API\)](#)
- [receive \(IPMT_API\)](#)
- [timeIndication \(IPMT_API\)](#)
- [txDone \(IPMT_API\)](#)

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

API Header	API Payload
------------	-------------

Notification Contents

4.1 advReceived

Description

The 'advReceived' notification is generated by the mote when it is in promiscuous listen state (see the [Mote States table](#)) and it receives an advertisement.

Payload

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

4.2 events

Description

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

**Payload**

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

4.3 receive

Description

Informs the application that a packet was received.

Payload

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

4.4 timeIndication

Description

Time notification is sent by mote in response to external trigger of TIMEn pin. The time indicated in the packet is that of detecting the high->low transition on the pin. Refer to the product datasheet for timing requirements related to the pin.

Payload

Parameter	Type	Enum	Description
uptime	INT32U	none	Time from last reset (seconds)
utcTime	UTC_TIME_L	none	UTC time at TIMEn trigger



asn	ASN	none	Absolute Slot Time at TIMEn trigger
asnOffset	INT16U	none	Time from the start of the ASN (microseconds)

4.5 txDone

Description

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

Payload

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



5 Definitions

5.1 Commands

The following commands may be sent to the mote:

Command	Value (hex)	Description
setParam	0x01	Set Parameter
getParam	0x02	Get Parameter
join	0x06	Join the network
disconnect	0x07	Disconnect from the network
reset	0x08	Reset mote
testRadioTx	0x0B	Start transmit radio test
testRadioRx	0x0C	Start receive radio test
clearNv	0x10	Clear non-volatile configuration storage
requestService	0x11	Request service
getServiceInfo	0x12	Get service information
openSocket	0x15	Open UDP socket
closeSocket	0x16	Close UDP socket
bindSocket	0x17	Bind UDP socket
sendTo	0x18	Send packet
search	0x24	Go into promiscuous listen mode

5.2 Parameters

The following parameters may be used in [setParam](#) and [getParam](#) commands

Parameter	Value (hex)	Persistent	Description
macAddress	0x01	Y	MAC address of the mote



joinKey	0x02	Y	Security join key
networkId	0x03	Y	Network identifier
txPower	0x04	Y	Transmit power
joinDutyCycle	0x06	Y	Join duty cycle
eventMask	0x0B		Event mask
moteInfo	0x0C		Mote information
netInfo	0x0D		Network information
moteStatus	0x0E		Mote status
time	0x0F		Time information
charge	0x10		Charge
testRadioRxStats	0x11		Statistics from radio test
otapLockout	0x15	Y	OTAP lockout
shortAddr	0x17		Mote's short address
ipv6Addr	0x18		Mote's IPv6 address
routingMode	0x1D	Y	Routing mode
pwrSrcInfo	0x1F	Y	Power Source Information
autoJoin	0x24	Y	Auto Join mode

5.3 Notifications

The following [notification](#) types may be generated by the mote:

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



5.4 Response Codes

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

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

5.5 Service Types

The following table lists types of services.

Type	Value (hex)	Description
bandwidth	0x00	Bandwidth-type service



5.6 Service States

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

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

5.7 Protocol Types

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

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

5.8 Packet Priorities

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

Priority	Value (hex)	Description
low	0x00	Low priority
medium	0x01	Medium priority
high	0x02	High priority

5.9 Mote States

The following table lists mote states

State	Value (hex)	Description
init	0x00	Initializing
idle	0x01	Idle, ready to be configured or join
searching	0x02	Searching for network
negotiating	0x03	Sent join request
connected	0x04	Received at least one packet from the Manager



operational	0x05	Configured by Manager and ready to send data
disconnected	0x06	Disconnected from the network
radiotest	0x07	The mote is in radio test mode
promiscuous listen	0x08	The mote received <i>search</i> command and is in promiscuous listen mode.

5.10 Events

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

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

5.11 Alarms

The mote may declare the following alarms:

Alarm	Value (hex)	Description
nvError	0x01	Detected an error in persistent configuration storage (NV)
otpError	0x04	Detected an error in calibration or bsp data in flash

5.12 Radio test types

The following tests may be performed via [testRadioTx](#) command:

Test type	Value (hex)	Description



packet	0x00	Packet transmission
cm	0x01	Continuous modulation
cw	0x02	Continuous Wave

5.13 Packet transmit status

Packet transmit status is returned as part of [txDone](#) notification.

Status	Value (hex)	Description
ok	0x00	Packet sent into the network
fail	0x01	Packet dropped