

SmartMesh IP Embedded Manager CLI Guide

Table of Contents

1	About This Guide	4
1.1	Related Documents	4
1.2	Conventions Used	6
1.3	Revision History	8
2	Introduction	9
2.1	CLI Access	9
2.1.1	Login/Logout	9
2.1.2	Managing user and viewer Passwords	10
2.1.3	Mote Commands	10
3	Commands	11
3.1	delete acl	11
3.2	delete mote	12
3.3	exec clearStat	13
3.4	exec exchJoinKey	14
3.5	exec exchNetId	15
3.6	exec restore	16
3.7	exec setAdv	17
3.8	exec setDnFrame	18
3.9	exec sendData	19
3.10	exec start	20
3.11	help	21
3.12	log	22
3.13	login	23
3.14	logout	24
3.15	onechan	25
3.16	ping	26
3.17	radiotest	27
3.17.1	radiotest on/off	27
3.17.2	radiotest tx	28
3.17.3	radiotest rx	30
3.17.4	radiotest stat	31
3.18	reset	32
3.19	set acl	33
3.20	set config	34
3.21	seti	36
3.22	show	37
3.22.1	show acl	39
3.22.2	show config & show curconfig	40
3.22.3	show mac	41

3.22.4	show mote	42
3.22.5	show motever	44
3.22.6	show path	45
3.22.7	show stat	46
3.22.8	show status	48
3.22.9	show time	50
3.22.10	show trace	51
3.22.11	show ver	52
3.23	showi	53
3.24	sm	54
3.25	su	55
3.26	trace	56
3.26.1	trace bw	58
3.26.2	trace fa	59
3.26.3	trace glbcmd	60
3.26.4	trace io (reserved)	61
3.26.5	trace iodata (reserved)	62
3.26.6	trace link	63
3.26.7	trace loop	64
3.26.8	trace monitor	65
3.26.9	trace motest	66
3.26.10	trace netmode	67
3.26.11	trace opt	68
3.26.12	trace power	69
3.26.13	trace rawio_enc (reserved)	70
3.26.14	trace route	71
3.26.15	trace spl_task & trace spl_ack (reserved)	72
3.26.16	trace stats	73
3.26.17	trace timeout	74
3.26.18	trace tpglock (reserved)	75

1 About This Guide

1.1 Related Documents

The following documents are available for the SmartMesh IP network:

Getting Started with a [Starter Kit](#)

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

User's Guide

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

Interfaces for Interaction with a Device

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

Software Development Tools

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

Application Notes

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

Documents Useful When Starting a New Design

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

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

Other Useful Documents

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


1.2 Conventions Used


The following conventions are used in this document:


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


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

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

 Tips provide useful information about the product.

 Informational text provides additional information for background and context

 Notes provide more detailed information about concepts.

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

`code blocks display examples of code`

The CLI commands are described using the following notations and terminology:

	Indicates alternatives for a field. For example, <code><moteld> #<MAC></code> indicates that you can specify a mote by its mote ID or MAC address.
< >	Indicates a required field.
{ }	Indicates a group of fields.
[]	Indicates an optional field.

MAC address	<p>When specifying a MAC address, do not use spaces. You may omit leading zeros and hyphens. In cases where the command syntax allows either the MAC address or mote ID to be specified, the MAC address must be preceded by the # symbol.</p> <p>The following examples are all valid:</p> <p>22CA</p> <p>00000000000022CA</p> <p>00-00-00-00-00-00-22-CA</p>
-------------	--

1.3 Revision History

Revision	Date	Description
1	07/17/2012	Initial Release
2	08/10/2012	Updated radiotest tx command
3	03/18/2013	Numerous small changes
4	10/22/2013	Minor corrections
5	04/04/2014	Updated and clarified radiotest commands;
6	10/28/2014	Clarified show motever command; Other minor changes
7	04/22/2015	Deprecated autostart command; Other minor changes
8	12/03/2015	Deprecated software licencing commands; Added show memory command; Renamed guide to focus on embedded manager
9	11/07/2016	Updated delete and set config commands; Added show mac and exec start commands

2 Introduction

This guide describes the commands that you can send to a SmartMesh IP Manager by logging on to its Command Line Interface (CLI). The CLI is available by connecting a serial terminal program to the Manager. The CLI is intended for human interaction with a manager, e.g. during development, or for interactive troubleshooting. Most commands are atomic - a command and its arguments are typed into the CLI, and a response is returned. For example, the `help` command returns a list of possible commands. Traces are not atomic - once started, they generate output asynchronously until cancelled.

For a machine-to-machine communications (e.g. a host program talking to the manager), the Application Programming Interface (API) is used. See the [SmartMesh IP Manager API Guide](#) for details on that interface.

2.1 CLI Access

There are two dedicated serial ports on the SmartMesh IP manager: one is for API communication with an external application, and the other is dedicated to this command line interface.

You can log on to the CLI from any serial terminal program (such as HyperTerminal or Tera Term):

- **Serial 0** — If connecting to an evaluation board integrated with an FTDI serial-to-usb interface, the CLI will be found on the 3rd COM port mapped onto your system.

The default serial port settings are 9600 baud, 8 data bits, No parity, 1 stop bit, no flow control.

2.1.1 Login/Logout

There are two sets of privileges on this system, namely *user* and *viewer*. The *user* privilege allows for system settings to be set and the *viewer* privilege only allows the viewing of manager and network information.

To login to the manager CLI from the terminal program enter either of the following usernames and passwords:

```
login viewer
```

```
login user
```

To logout of the Manager CLI:

```
logout
```

2.1.2 Managing user and viewer Passwords

The default passwords should be changed with the following commands (after logging in with "user" privileges):

```
set config pwdviewer <newpassword>

set config pwduser <newpassword>
```

2.1.3 Mote Commands

Commands beginning with an 'm' such as `mtrace` or `minfo` are specific to the Access Point "mote" and are described in the [SmartMesh IP Mote CLI Guide](#) documentation.

3 Commands

This manual describes the CLI commands available in the SmartMesh IP manager. The CLI is case-insensitive. In most cases a command will be recognized by the shortest unambiguous string, so the following are all equivalent:

```
> trace rawio_enc on  
  
> trace rawio on  
  
> trace raw on
```

3.1 delete acl

Description

The `delete acl` command removes a mote's entry in the access control list (ACL), clearing the mote's join key and join counter. Deleting a mote from the ACL prevents the mote from joining the network, or rejoining if the mote is already in the network. It does not force a mote to leave the network. If all motes are deleted from the ACL, the system reverts to using a common join key. This change is persistent.

Syntax

```
delete acl <macAddress | all>
```

Parameters

Parameter	Description
macAddress	MAC address of the mote or 'all'

Example

```
delete acl 00-17-0D-00-00-38-00-21  
  
delete acl all
```

3.2 delete mote

Description

Delete a mote from the manager. This command will not remove an operational mote from the network. It is merely intended as a way to remove it from the list of motes known to the manager. Note that you can only delete a mote from the network if the mote is in the **Lost** or **Idle** state, or was used as a **Blink** mote. This change is persistent.

This command should be used to clear a mote's join counter in the event that it is completely reflashed. Failure to do so will prevent a mote from joining the network until the join counter matches. The same applies to *blink* packets when using that mode, none will be delivered until the join counter matches.

Syntax

```
delete mote <moteId | mac>
```

Parameters

Parameter	Description
moteld mac	ID or mac address of the mote to be deleted from the network.

Example

```
delete mote 2  
  
delete mote 00-17-0D-00-00-60-06-11
```

3.3 exec clearStat

Description

Clear all network statistics.

Syntax

```
exec clearStat
```

Parameters

Parameter	Description
-----------	-------------

Example

```
exec clearStat
```

3.4 exec exchJoinKey

Description

Replace the join key for a specified mote. The message is sent to the mote and is also changed in the ACL. This change is persistent.

Syntax

```
exec exchJoinKey <address> <joinKey>
```

Parameters

Parameter	Description
address	Mote ID or MAC address of mote to be changed
joinKey	16-byte join key

Example

```
exec exchJoinKey 00-17-0D-00-00-38-00-21 000102030405060708090A0B0C0D0E0F
```

3.5 exec exchNetId

Description

Exchange the Network ID. This command will change the Network ID of the manager and all motes connected to the network. The new Network ID takes effect the next time the network is restarted. Network IDs 0 and 65535 are reserved and should not be used. This change is persistent.

Syntax

```
exec exchNetId <netId>
```

Parameters

Parameter	Description
netId	Integer between 1 and 65534

Example

```
exec exchNetId 100
```

3.6 exec restore

Description

Restores all factory default settings. This change is persistent.

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

Syntax

```
exec restore
```

Parameters

Parameter	Description
-----------	-------------

Example

```
exec restore
```


3.7 exec setAdv

Description

This command controls advertising in the network. Setting to *off* turns off all advertisements and saves the most power, but motes will not be able to hear and join the network in this state. Setting to *on* turns on advertisements to allow motes to join the network.



It is dangerous to turn off advertising in the network. When advertising is off, new motes can not join and existing motes can not rejoin the network after a reset. Turning off advertising may be useful in unusual situations, such as to prevent motes from joining the network or to save power. In most cases, it is best to allow advertising to remain under the control of the manager.

Syntax

```
exec setAdv <on|off>
```

Parameters

Parameter	Description
state	on = turn on advertisements, off = turn off advertisements

Example

```
exec setAdv on
```

3.8 exec setDnFrame

Description

Change the downstream superframe size. This is only valid when running a network with the downstream superframe multiplier configuration parameter (*dnfr_mult*) set to either 2 or 4. This command will turn that multiplier on (normal), or off (fast). Using this command disables the automatic switching of superframe size after the network has formed. As an example, if the starting frame size for the network is 256 slots and *dnfr_mult* is set to 4, then changing to 'normal' multiplies the superframe by 4 to a 1024-slot superframe.

Syntax

```
exec setDnFrame <mode>
```

Parameters

Parameter	Description
mode	'fast' or 'normal'

Example

```
exec setDnFrame normal
```

3.9 exec sendData

Description

Send packet with specified payload to a mote. This CLI command is equivalent to invoking API command *sendData*.

Syntax

```
exec sendData <destination> <srcPort> <destPort> <priority> <payload>
```

Parameters

Parameter	Description
destination	Destination of the packet. Can be specified as either Mote ID or MAC address
srcPort	UDP source port of the packet
destPort	UDP destination port of the packet
priority	Priority of the packet. 0(low)-2(high)
payload	Payload bytes of the packet, in hex. Maximum size of payload is 40 bytes


Example

Send packet to Mote ID=2, source port=20, destination port=20, priority=1, payload bytes="0x11,0x22,0x33,0x44,0x55":

```
exec sendData 2 20 20 1 1122334455
```

3.10 exec start

Description

 This command has been deprecated and should not be used.

The *exec start* command tells the manager to allow the network to start forming (begin accepting join requests from devices).

3.11 help

Description

Show help. Entering this command without parameters displays the list of all available commands to the current user and mode. Help on a specific command may be obtained by entering that command as an argument.

Syntax

```
help [command]
```

Parameters

Parameter	Description
command	Any of the CLI commands

Example

```
help
```

3.12 log

Description

Retrieves debug log information from a mote's flash memory. This is used for debugging if a mote reset from a network.

Syntax

```
log <moteId>
```

Parameters

Parameter	Description
moteld	ID of the mote of interest

Example

```
log 2
```

3.13 login

Description

The CLI interface requires a login, and the password entered determines the privilege used for the session. The default passwords match the two privilege levels: `viewer` and `user`. The `viewer` cannot make any configuration changes to the manager. The `user` has access to all commands. The `login` command can be used repeatedly without logging out to switch between privilege levels. Passwords for the two privilege levels can be changed using the `set config` command.

Syntax

```
login [<user>:] <password>
```

Parameters

Parameter	Description
<code>user</code>	<code>viewer</code> or <code>user</code>
<code>password</code>	password for the privilege level. default passwords as shipped are "user" for <code>user</code> and "viewer" for <code>viewer</code> note: passwords can be changed with the <code>set config</code> command

Example

```
login user: me$h  
  
login me$h
```

3.14 logout

Description

Logout from the current CLI session.

Syntax

```
logout
```

Parameters

Parameter	Description
-----------	-------------


Example

```
logout
```


3.15 onechan

Description

Setup the network to run on a single channel. This command may be used for RF compliance testing. This command takes effect only after system reset. This setting is persistent, and must be turned off to revert to normal operation.

 Channel numbering for this command is 0-15, corresponding to IEEE 2.4 GHz channels 11-26

Syntax

```
onechan <channel>
```

Parameters

Parameter	Description
channel	Channel (0-15) on which to operate or 'off'

Example

```
onechan 2  
onechan off
```

3.16 ping

Description

Request a reply from a mote in the network. Mote should respond with temperature and voltage. Note this is a Dust command - it does not use ICMP echo.

Syntax

```
ping <moteId>
```

Parameters

Parameter	Description
moteld	ID of the mote to be pinged.

Example

```
> ping 2
Sending ping request to mote 2
> Ping response from mote 2, time=350 msec v=3603 t=26
```

3.17 radiotest

3.17.1 radiotest on/off

Description

Enable or disable radiotest mode on the device. Radiotest functionality can be used to exercise the radio for certification and testing purposes. This command takes effect after reboot and the selected mode persists until changed, i.e. if ON, it will remain on even after reset or power cycle until the mode is set to OFF and the device is rebooted.

Syntax

```
radiotest <mode>
```

Parameters

Parameter	Description
mode	on - put device into radiotest mode after reboot off - put device into normal master mode after reboot

Example

Put device into radiotest mode:

```
radiotest on
```

Return device to normal operational mode:

```
radiotest off
```

3.17.2 radiotest tx

Description

The `radiotest tx` command allows the user to initiate a radio transmission test. This command may only be issued in radiotest mode. Three types of transmission tests are supported:

- pk - Packet Transmission
- cm - Continuous Modulation
- cw - Continuous Wave (unmodulated signal)
- pkcca - Packet transmission with clear channel assessment (CCA) enabled (Available in IP Manager \geq 1.3.0 and IP mote \geq 1.4.0)


In a packet transmission test, the device generates a *repeatCnt* number of packet sequences. Each sequence consists of up to 10 packets with configurable sizes and delays. Each packet consists of a payload of up to 125 bytes, and a 2-byte 802.15.4 CRC at the end. Byte 0 contains sender's stationId. 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 device generates continuous pseudo-random modulated signal, centered at the specified single channel. The test is stopped by resetting the device.

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

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

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

 stationId is available in SmartMesh IP Mote \geq 1.4, SmartMesh IP Manager \geq 1.3.0, SmartMesh WirelessHART mote \geq 1.1.2

Syntax

```
radiotest tx <testType> <chanMask> <power> [<stationId> <repeatCnt> {<pkLen><delay>...}]
```

Parameters

Parameter	Description
testType	Type of transmission test to initiate: 'pk' = packets, 'cm' = continuous modulation, 'cw' - continuous wave, "pkcca" = packets with CCA.
chanMask	Hexadecimal bitmask of channels (0–15) for the test. Bit 0 corresponds to channel 0. For continuous wave and continuous modulation tests, only one channel should be enabled.
power	Transmit power, in dB. Valid values are 0 and 8.
stationId	Unique (0-255) station id of the sender. Must match station id value of the receiver.
repeatCnt	Number of times to repeat the packet sequence (0=do not stop). Applies only to packet transmission tests.
pkLen	Length of packet (2-125 bytes)
delay	Delay after transmission (0-65535 microseconds)

Example

Initiate packet test on channels 0,1 (chMap=0x03), with output tx power of 0 dBm, station id = 26

Repeat the sequence 5 times: 50-byte packet, 20ms delay, 30-byte packet, 20msec delay

```
radiotest tx pk 0x3 0 26 5 50 20000 30 20000
```

Start transmission with continuous modulation on channel 0 with output tx power of 8 dB

```
radiotest tx cm 0x1 8
```


Start transmission with continuous wave on channel 1 with output tx power of 8 dB


```
radiotest tx cw 0x2 8
```

3.17.3 radiotest rx

Description

The `radiotest rx` command puts the radio into receive mode where statistics on packet reception are collected. The nonzero station id specified must match station id of the sender, which is necessary to isolate traffic of multiple tests running in the same radio space. Statistics may be viewed with the `radiotest stat` command.

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

 `stationId` is available in SmartMesh IP Mote \geq 1.4, SmartMesh IP Manager \geq 1.3.0, SmartMesh WirelessHART mote \geq 1.1.2

Syntax

```
radiotest rx <chanMask> <time> <stationId>
```

Parameters

Parameter	Description
chanMask	Hexadecimal bitmask of channels (0–15) for the test. Bit 0 corresponds to channel 0. Only a single channel may be specified for this command.
time	Duration of receive test, in seconds. 0=do not stop
stationId	Unique (1-255) id of the receiver. Must match sender's station id. Station id 0 may be used to accept packets from any sender.

Example

Put device into receive mode for 60 seconds on channel 2, use station id 26:

```
radiotest rx 0x4 60 26
```

3.17.4 radiotest stat

Description

The `radiotest stat` command displays packet reception statistics collected during the previously run `radiotest rx` command. This command may only be used when the device is in radiotest mode.

Syntax

```
radiotest stat
```

Parameters

Parameter	Description
-----------	-------------

Example

```
>radiotest stat
Radio Test Statistics
  OkCnt   : 0
  FailCnt : 0
```

3.18 reset

Description

Reset a specified entity in the network: either a mote or the manager. This command requires *user* privilege.

Syntax

```
reset <entity>
```

Parameters

Parameter	Description
entity	<ul style="list-style-type: none">• When called with "system", resets the manager and by extension the entire network.• When called with "mote", mote can be referenced by Mote ID or MAC address.

Example

```
reset mote 2  
  
reset system
```


3.19 set acl

Description

The `set acl` command is used to create a new ACL entry or change an existing one. The key is in hexadecimal with two characters per byte. This change is persistent. The maximum number of entries is 1,200.

Syntax

```
set acl mac=<macAddr> key=<joinKey>
```

Parameters

Parameter	Description
macAddr	MAC address of the mote
joinKey	16-byte (32 character) join key

Example

```
set acl mac=01-23-45-67-89-AB-CD-EF key=000102030405060708090A0B0C0D0E0F
```

3.20 set config

Description

Set a configuration parameter in the Manager. The change will take effect upon the next system start (after reset or power cycle). This change is persistent. Note that features that require a license will take two resets - once for the license to take effect (and enable the settings change), and once for the setting to take effect.



The license field has been deprecated in Manager \geq 1.3.0 .There is no need to use a license to enable > 32 mote networks.

Syntax

```
set config <param>=<value>
```

Parameters

Parameter	Description
-----------	-------------

param	<ul style="list-style-type: none"> • <i>netid</i>: network id • <i>txpower</i>: radio tx Power • <i>frprofile</i>: frame profile id • <i>maxmotes</i>: maximum number of motes with AP (1-101: depends on manager part number and installed license) • <i>basebw</i>: base bandwidth (0 or >=10) • <i>dnfr_mult</i>: downstream frame multiplier (1,2,4) • <i>numparents</i>: minimum number of parents (1-4) • <i>cca</i>: CCA value (0,1,2,3) • <i>channellist</i>: bitmap of channels in the whitelist to use for communication, all others blacklisted. Bit 0x0001 corresponds to channel 0 and bit 0x8000 corresponds to channel 15. (0=not used, 1=used). See the SmartMesh IP User's Guide section "Channel Blacklisting" for restrictions on the number of channels. • <i>autostart</i>: deprecated - do not use. • <i>locmode</i>: reserved • <i>bbmode</i>: backbone mode (0=off,1=up,2=bidirectional) • <i>bbsize</i>: backbone frame size (if bbmode=1, bbsize=1,2,4,8. If bbmode=2, bbsize=2) • <i>license</i>: license (hex, e.g. 00-12-34...) • <i>pwdviewer</i>: 'viewer'-level password • <i>pwduser</i>: 'user'-level password • <i>commonjoinkey</i>: common join key • <i>ip6prefix</i>: IPv6 address prefix • <i>ip6mask</i>: IPv6 mask • <i>radiotest</i>: radio test mode (0=off,1=on) • <i>bwmult</i>: over-provision by value/100 (100-1000) • <i>onechannel</i>: channel for single channel network (0-14,0xFF)
value	Value (refer to the configuration parameter table)

Example

```
set config netid=100
```

3.21 seti

Description

The `seti` command is used change an internal INI parameter. This command is for advanced configuration as instructed by an application note, and requires `superuser` privileges (see `su`).

Syntax

```
seti ini <param>[= | ' ']<args>
```

Parameters

Parameter	Description
parameter	Parameter to be changed. Parameters that can be changed to modify manager behavior to achieve a particular outcome are described in various application notes
args	Arguments to the specified parameter

Example

```
seti ini iscascading 1
```

3.22 show

Description

Shows status information for various system objects.

Syntax

```
show <object> [args]
```

Parameters

Parameter	Description
args	see object table for possible arguments

object	Object	Description	Arguments
	acl	Access control list	
	curconfig	Currently used config parameters	
	config	Persistent config parameters	
	mote	Mote status	Can be called with -a or -l, <moteID macAddr *>
	motever	Mote version	<moteID macAddr>
	path	Path information	
	stat	Statistics (reliability, stability, etc.)	
	status	Network status	
	time	Current time in UTC (if set) and ASN	
	trace	Active traces	
	ver	Manager version information	
	memory	Manager memory configuration. Type 0=primary RAM, type 1= external RAM, type 2 = 2k secondary internal RAM. Command available in manager >= 1.2.0	

Example

```
show mote 2
```

3.22.1 show acl

```
> show acl
ACL:

    MAC: 00-17-0D-00-00-38-FF-FF

    MAC: 00-17-0D-00-00-37-6E-A1
```

This command shows the nodes currently whitelisted on the manager Access Control List.

3.22.2 show config & show curconfig

```
> show config
netid = 302
txpower = 8
frprofile = 1
maxnotes = 33
basebw = 9000
dnfr_mult = 1
numparents = 2
cca = 0
channellist = 00:00:7f:ff
autostart = 1
locmode = 0
bbmode = 0
bbsize = 1
license = 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
ip6prefix = fe:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
ip6mask = ff:ff:ff:ff:ff:ff:ff:ff:00:00:00:00:00:00:00:00
radiotest = 0
bwmult = 300
onechannel = 255
```

Both of these commands return the same data structure. The `show config` command will display the persistent parameters, i.e. the ones used after the next boot. The `show curconfig` command displays the current parameters being used. See [set config](#) for a description of parameters.

3.22.3 show mac

```
> show mac
MACs:
----
```

RESERVED: DO NOT USE. This command is an internal command that should not be used.

3.22.4 show mote

```

> show mote 2
Mote #2, mac: 00-17-0D-00-00-38-16-6B
  State: Oper, Hops: 1.1, Uptime: 0-00:30:39, Age: 1
  Regular. Route/TplgRoute.
  Power Cost: Max 65534, FullTx 110, FullRx 65
  Capacity links: 200, neighbours 31
  Number of neighbors (parents, descendants): 8 (2, 15)
  Bandwidth total / mote exist (requested): 90 / 954 (987)
    Links total / mote exist (requested): 64.0 / 6.0 (5.8)
    Link Utilization : 1.0
  Number of total TX links (exist / extra): 64 / 0
  Number of links : 130
    Compressed : 5
    Upstream tx/rx : 122 (64/58) (Rx10=58.0)
    Downstream rx : 3
  Neighbors:
    -> # 1 Q: 71% RSSI: -59/0
    -> # 7 Q: 55% RSSI: -55/-49
    <- # 9 Q: 97% RSSI: -39/-40
    <- #15 Q: 94% RSSI: -37/-39
    <- #17 Q: 87% RSSI: -40/-43
    <- #18 Q: 91% RSSI: -44/-44
    <- #20 Q: 29% RSSI: -76/-74
    <- #24 Q: 29% RSSI: -76/-75

```

Description of fields:

- Mote: Short address of mote
- mac: EUI-64 of mote
- State: Manager-assigned current state for the mote, one of **Idle**, **Negot1-2**, **Conn1-5**, **Oper**, or **Lost**
- Hops: The average number of hops taken by this mote's upstream packets, as measured by the TTL when received at the AP
- Uptime: Time since the mote's most recent state change
- Age: Time, in seconds, since the manager received the most recent upstream packet from this mote
- Power type: Power (maxStCurrent in powerSrcInfo param = 0xffff), Regular, or Low Power (maxStCurrent less than needed for routing)
- Route type reported by the mote: Route, No-route (from routingMode param)
- Route type as assigned by the manager: TplgRoute, TplgNo-Route
- Power Cost: powerSrcInfo parameters reported by mote during joining
- Number of neighbors: first entry is # parents + # children = # nbrs, first entry in parentheses is # parents, second entry in parentheses is descendants. From this # children = # nbrs - # parents

- Bandwidth (ms/packet): the bandwidth section is devoted to upstream traffic and upstream links only; the *total* shows the combination of mote-local and descendant traffic; the *mote exist* value shows the bandwidth that the mote itself is responsible for; the *requested* value in parentheses shows the bandwidth that the mote asked for through service requests. Lower values here represent more bandwidth. In general, the *mote exist* value will be slightly less than the *requested* value since there is some roundoff as the manager cannot add fractional links.
- Links (links/superframe): the *total* shows the combination of links added to support mote-local and descendant traffic; the *mote exist* value shows the number of links added specifically for this mote's service requests; the *requested* value in parentheses shows a floating point number of links that could support the traffic, so the *mote exist* value is the *requested* value rounded up to the nearest integer.
- Link Utilization: a number between 0 and 1 representing how close to the provisioning limit the manager thinks the mote is. For example, if the manager expects the mote to send 1 pkt/s, the provisioning is 3x, and the mote has 10 link/s upstream, the utilization would be $1 * 3 / 10 = 0.3$. This value is used to scale traffic requirements to the mote's parent to keep link numbers down in low-traffic networks.
- Number of total TX links: the *exist* value shows how many upstream links are currently assigned, and the *extra* value shows how many will be deleted during the next optimization cycle
- Number of links: Total links across all slotframes, not just the upstream links; also this is the sum of the next three rows
- Compressed: Number of compressed links, which are used for advertising, join listen, and discovery; these are assigned during the mote's join and never changed, so the manager saves memory by storing them in a compressed format
- Upstream: Total, (# Tx / # Rx as an integer), (# Rx needed, not rounded up)
- Downstream: # Rx links from parents; note that 1-hop motes may have two downstream Rx links from the AP, one broadcast and one multicast
- Neighbors: relationship (-> to parent, <- to child, - discovered), neighbor's mote ID, path quality (30% or 74% until path stability is measured), RSSI as measured by this mote, RSSI as measured by the neighbor mote

Note that "routing type" can be set either on the mote or on the manager. If either the mote or the manager declares a mote to be non-routing, then the mote will not be assigned children or advertisement links.

3.22.5 show motever

```
> show motever 2
Mote #2, mac: 00-17-0D-00-00-38-06-28. Ver SW: 1.1.0-29 HW: 1.33
Mote #2, mac: 00-17-0D-00-00-38-06-28. Vendor: 1. App ID/Ver: 1/1.1.0-29
```

This command displays all version information for a mote in the network. Note that this is a command sent to the mote over the air and the response will not be instantaneous.

Two responses will be displayed. The first is the Network stack software version, and the second will contain the vendor ID and application layer software version.

3.22.6 show path

```
> show path
MoteID * {NeighborId:{P|C|-}:Quality(Src)}...
  P - neighbor is parent, C - neighbor is child, - - no parent relationship
  Quality - quality of path (100-best, 0-worst)
  Src - source for quality calculation (s-statistics, r-RSSI)
Network Average Quality: 78
Paths:
1 * 2:C:97(s) 3:C:74(r)
2 * 1:P:97(s) 3:C:74(r)
3 * 1:P:74(r) 2:P:74(r)
```

This command displays all the paths that the manager currently knows about in the network whether they have active links or not. Each mote gets one row of the display. By looking at the second row, we can see the paths associated with Mote ID 2. This row indicates that mote 1 is a (P)arent, and this path 2 to 1 has a Quality which has been measured based on success/fail (s)tatistics to be 97% - this is also called the "stability" of the path. The report also indicates that mote 3 is a (C)hild and that there is not yet any stability measurement on this path so we are using (r)SSI to estimate the quality.

3.22.7 show stat

```

> show stat

Manager Statistics -----
established connections: 1
dropped connections    : 0
transmit OK           : 0
transmit error        : 0
transmit repeat       : 0
receive OK            : 0
receive error         : 0
acknowledge delay avrg : 0 msec
acknowledge delay max  : 0 msec

Network Statistics -----
reliability: 100% (Arrived/Lost: 7217/0)
stability:   99% (Transmit/Fails: 14304/204)
latency:     200 msec

Motes Statistics -----
Mote   Received   Lost  Reliability Latency Hops
#2      257         0    100%        580   1.3
#3      249         0    100%        450   1.0
#4     6463         0    100%        250   2.4
#5      248         0    100%        150   1.2

```

This command displays the mote and network statistics. The counters here are incremented or averaged over the lifetime of the network, or since they have actively been cleared using an `exec clearStat` CLI command or a `clearStatistics` API command.

In this example, mote 4 is generating data much more often than the other motes. It is also at 2.4 hops, so all its packets are being forwarded through other motes. Mote 3 is at 1.0 hops, so all its packets are going straight to the AP (it is the "single parent" mote). Motes 2 and 5 are forwarding a fraction of their traffic through mote 3.

Analyzing the numbers:

Reliability: Arrived = 7217 = 257 + 249 + 6463 + 248, Lost = 0. The arrived/lost counters and reliability are kept real-time on the manager so will always be up-to-the-second accurate.

Stability: Transmit = 14304 ≈ 7217 (the packets that arrived) + 204 (the packets that failed and needed to be retransmitted) + 6463 (the packets from mote 4 that needed to be forwarded by 2, 3, & 5) + 50 (20% of mote 5's packets forwarded by mote 3) + 77 (30% of mote 2's packets forwarded by mote 3) + 293 (packets from mote 4 that went to motes 2 and 5 and needed to be forwarded through mote 3). The stability counters and average stability are calculated based on mote health reports so they lag the reliability statistics by up to 15 minutes.

Manager Statistics

- *established connections*: Number of API connections made
- *dropped connections*: Number of API connections dropped
- *transmit OK*: API packets (e.g. notifications) acknowledged by the client application
- *transmit error*: API packets not acknowledged
- *transmit repeat*: API packets retried (may be > transmit error if multiply retried)
- *receive OK*: API commands that the manager acknowledged
- *receive error*: API commands that the manager could not acknowledge (due to framing or other errors)
- *acknowledge delay avrg*: Average time to receive a response on an API packet
- *acknowledge delay max*: Maximum time to receive a response on an API packet

3.22.8 show status

```

> show status
S/N:      00-17-0D-00-00-38-07-13
MAC:      00-17-0D-00-00-38-07-13
IP6:      FE:80:00:00:00:00:00:00:17:0D:00:00:38:07:13
HW ver:   1.33
NetworkID: 302
Base bandwidth (min links): 9000 (1)
Frame size Up/Down 259/259. Number of working timeslots 256.
Available channels 15
Base timeslot(TSO) 235
Network mode is Steady State
Downstream frame 259 timeslots
Optimization is On
Advertisement is On
AP output CTS is Ready
Backbone is off
Location is off
API is Connected
License 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
Network Regular Mode
Total saved links 28
One channel mode: OFF.

```

This command displays some fixed global details about the network followed by several states which may change over the network lifetime.

- S/N: EUI-64 of Manager
- MAC: Long address of manager (may not match EUI-64)
- IP6: IPv6 address. Shown here using link-local prefix
- HW ver: Hardware version
- NetworkID: 0-65535
- Base Bandwidth: Global bandwidth target for each mote, in ms
- Frame size: Randomized 256-284 slots. Not all values are possible. Working slots is always 256 and represents the number of slots in the slotframe that can have active links
- Available Channels: 15 unless blacklisting
- Base Timeslot: This offsets all links by a random number between 0 and 255 to improve performance when multiple networks are co-located
- Downstream Frame: Set by *dnfr_mult* config parameter, this can either be 1x, 2x, or 4x the frame size above
- Optimization: On by default
- Advertisement: On by default. Can be enabled/disabled through `exec setAdv on` or `exec setAdv off`
- AP output: AP is ready to accept packets
- Backbone: Off by default. Can be enabled through *bbmode* config parameter

- Location: reserved
- API: Whether there is a device connected to the API
- License: Key
- Network: Mode
- Total saved links: links manager is storing, this is limited to 500 because of memory constraints
- One channel mode: Used for testing. Enabled one channel testing on channel x by `set config onechannel x`.
Disable by `set config onechannel 255`.

3.22.9 show time

```
> show time
Current UTC:      1025666459.752 sec
Current second:  1260 sec
Current OS time: 1260438 ticks
Current ASN:     173759 slots
```

This command displays three different measures of time: UTC, the system time (in ms ticks), and the current ASN for the network.

3.22.10 show trace

```
> show trace
  io
  motest
```

This command displays the currently active manager traces. In this example, the input-output and mote state traces are active.

3.22.11 show ver

```
> show ver  
SmartMesh IP Manager ver 1.1.0.25.
```

This command displays the version of code running on the manager.

3.23 showi

Description

Shows status information for various internal system objects. Intended for debugging during Linear manager development (i.e. it is not intended to be used by end customers or system integrators)

Syntax

```
showi <object>
```

Parameters

Parameter	Description
object	<i>ini</i> : System INI parameters (cannot be changed) <i>linkmap</i> : Bitmap of assigned links <i>loopfinder</i> : Loop finder tables <i>memory</i> : Memory pool information <i>stacks</i> : Tasks stack usage <i>toplock</i> : Topology Lock statistics

Example

```
showi ini
```

3.24 sm

Description

Show motes in the network

Syntax

```
sm [-v]
```

Parameters

Parameter	Description
-v	verbose

Example

```
> sm

      MAC                MoteId State Nbrs Links Joins Age StateTime
00-17-0D-00-00-38-07-13    1 Oper   2   25   1   0  0-00:45:39
00-17-0D-00-00-38-06-28    2 Oper   2   15   1   1  0-00:45:21
00-17-0D-00-00-38-04-2F    3 Oper   2   15   2   1  0-00:28:49
Number of motes (max 33): Total 3, Live 3, Joining 0
```

This command lists all the motes currently or previously in the network.

- MAC: EUI-64 of the mote
- MoteID: short address assigned to this mote by the manager. MoteID 1 is always the AP.
- State: Current state of each mote (Negot, Conn, Oper, Lost)
- Nbrs: Number of neighbors with which this mote has active links.
- Links: Total number of links, compressed and normal.
- Joins: Shows how many times the mote has advanced to the Operational state.
- Age: Seconds since the most recent packet was received by the manager from this mote.
- StateTime: Time (d-hh:mm:ss) since the mote was advanced to its current state. When a mote is Operational, StateTime shows how long the mote has been in the network.

3.25 su

Description

The `su` command is used to change the current user. This command is typically used to move from `viewer` to `user` credentials.

Syntax

```
su user
```

Parameters

Parameter	Description
password	Password for the privilege level. Default passwords as shipped are "user" for <code>user</code> and "viewer" for <code>viewer</code> . Note: passwords can be changed with the <code>set config</code> command The special <code>superuser</code> password "becareful" can be used to enable the <code>seti</code> command.


Example

```
su user
```

3.26 trace

Description

Turn traces on or off. This command is used for evaluation and debug purpose and is only available to the *user* login level. Traces should not be left running on a network in the field unless the user is trying to diagnose a problem. Some reserved traces require special manager configuration. If called with no arguments, returns current state of all traces

 Some traces can generate enough activity, particularly in larger networks, that they can negatively impact manager performance and may result in corrupted prints at the default CLI baud rate. These traces will return an error indicating that the trace "is not supported for this CLI baud rate." They can be enabled without changing the CLI baud rate by issuing the following commands before attempting to enable the trace. This setting is not persisted through reset.

```
> su becareful
> debug hscli on
```

Syntax

```
trace [<module> <mode>]
```

Parameters

Parameter	Description
-----------	-------------

module	<i>bw</i> : bandwidth / services <i>glbcmd</i> : global unicast commands <i>io</i> : manager rx/tx traffic (reserved) <i>iodata</i> : application rx/tx traffic (reserved) <i>loc</i> : reserved <i>link</i> : creation/deletion of links <i>monitor</i> : monitor events <i>motest</i> : mote state changes <i>netmode</i> : network mode <i>opt</i> : optimization <i>rawio_enc</i> : encrypted input-output packet (reserved) <i>power</i> : power cost calculation <i>route</i> : mote route calculation <i>spl_task</i> : serial events (reserved) <i>spl_ack</i> : serial ack timeouts (reserved) <i>stats</i> : statistics calculations <i>timeout</i> : reliable message timeouts <i>tplglock</i> : topology (reserved) <i>fa</i> : flash array <i>loop</i> : loopfinder <i>all</i> : enable all traces
mode	<i>on</i> : enable traces for the module <i>off</i> : disable traces for the module

Example

```
trace motest on
```

3.26.1 trace bw

```
>trace bw on  
3031355 : New Bandwidth Request. BW: 30000 msec. Links 1  
3060627 : New Bandwidth. #2 BW: 5631 msec. Links(req/real/extra) 1/1/0
```

This trace shows all new services requested by motes. There is one print when the service request is received at the manager and one when the change has been completed. In this example, the mote still has enough bandwidth to send a packet every 5631 msec.

3.26.2 trace fa

```
> trace fa on
```

```
1790561 : Flash Array. Save Data. Record #22 File 2mngarr1.bin. Offset 792
```

This trace prints all interactions the manager has with flash memory. The file is specified as well as the memory location.

3.26.3 trace glbcmd

```
> trace glbcmd on

2124787 : GlbCmd Send Timeout: 30 sec
2124788 : GlbCmd 1 Request for Send
2124789 : GlbCmd New Command 1 Execute Time: 1 sec
2124790 : GlbCmd 1 Send command to mote #1
2124799 : GlbCmd NETID. NetId: 200
Start Global Unicast Command exchNetId
2124829 : GlbCmd 1 Received ack from mote #1
2154788 : GlbCmd 1 Request for Send
2154789 : GlbCmd 1 Send command to mote #2
2154798 : GlbCmd NETID. NetId: 200
2155720 : GlbCmd 1 Received ack from mote #2
2184788 : GlbCmd 1 Request for Send
2184790 : GlbCmd 1 Send command to mote #3
2184798 : GlbCmd NETID. NetId: 200
2185829 : GlbCmd 1 Received ack from mote #3
2185830 : GlbCmd 1 finishes
```

This trace displays the process for sending any global (all motes) downstream commands. In the example, the manager exchanges the NetId for the AP and the two motes in the network. Global commands are sent reliable unicast so we can see each mote respond before the command terminates.

3.26.4 trace io (reserved)

```
> trace io on  
  
732985 : TX  ie=32:2e mesh=1223 ttl=127 asn=35486 gr=2 dst=10 rt=r8 r=4:7:8:9:10 tr=r:req:10; sec=s  
ofs=157 nc=0 ie=fe:00 IP=7e77 UDP=f7 sP=f0b0 dP=f0b0; cmd=GET_REQ_SRV len=3 type=0;
```

The “input-output” trace shows command traffic to and from the manager. This will not display application data. Contains a parsed net and IP headers, and payload information

- OS Timestamp (ms since the manager booted)
- TX is a downstream packet being sent by the manager, RX is an upstream packet being received by the manager
- ie: 15.4 information element descriptor, here header IE
- mesh: mesh dispatch
- TTL: decrementing mesh TTL counter, for received (RX) packets the hops can be calculated as 127-ttl
- ASN: ASN when packet was generated
- gr: graph on which the packet will travel, typically 2 for downstream or 1 for upstream
- dst: destination mote
- rt: route type, here a maximum 8-hop source route
- r: the explicit source route by-hop not including the manager or the first hop
- tr: transport header - here (r)eliable (req)uest with counter = 10
- sec: security session type - it will be "j" for the first joining packets using the join key and "s" using the session key afterwards
- ofs: encryption offset
- nc: nonce counter (there is currently a bug that displays this as always zero)
- ie: 15.4 IE descriptor, here terminator
- IP: Compressed IP header
- UDP: Compressed UDP header
- sP: Source port
- dP: Destination port
- cmd: network stack command
- len: command length

In this example, the manager is requesting a service report from mote 10.

3.26.5 trace iodata (reserved)

```
> trace iodata on
3508141 : RX  ie=32:1e mesh=4002 ttl=126 asn=24959 gr=1 src=3 rt=g tr=u:req:0; sec=s ofs=23 nc=208
ie=fe:00 IP=7e77 UDP=f7 sP=f0b9 dP=f0b9;
```

The “input-output data” trace shows application data traffic to the manager. This will not display command traffic. In this example, the manager received a data packet from mote 3. See [trace io](#) for parsing information.

3.26.6 trace link

```
> trace link on  
  
3628489 : Link Add      #2 -> #1 (1:211:0)  
3628491 : Link Add      #2 -> #1 (1:83:6)  
3628493 : Link Add      #2 -> #1 (1:243:5)  
3628495 : Link Add      #2 -> #1 (1:179:2)  
3628498 : Link Add      #2 -> #1 (1:115:7)
```

This trace shows all link additions and deletions decisions made by the manager. This will be followed by the actual commands to the motes to make the changes - these will be visible if `trace io` is on. The mote pair for the link is shown followed by the slotframe:slot:offset for the link.

3.26.7 trace loop

```
> trace loop on  
  
1657824 : Loopfinder. Connect 2 -> 6  
1660018 : Loopfinder. Disconnect 2 -> 3  
1660019 : Loopfinder. Recalculation
```

This trace shows the individual operations that the manager makes related to the loop checker. When two new motes are connected in a child-parent relationship, it is easy to fill in the new entries in the parent and ancestor tables. When two motes are disconnected, a full recalculation of the tables needs to be made.

3.26.8 trace monitor

```
> trace monitor on  
  
3722406 : Monitor event 'waitq'  
3722407 : Monitor event 'waitq' finish. Res 0  
3723409 : Monitor event 'waitq'  
3723409 : Monitor event 'waitq' finish. Res 0
```

This trace shows all events as they happen in the manager code. It is intended only for debugging purposes.

3.26.9 trace motest

```
> trace motest on
84410 : Mote #2 State:  Idle -> Negot1
86139 : Mote #2 State: Negot1 -> Negot2
88329 : Mote #2 State: Negot2 -> Conn1
90185 : Mote #2 State: Conn1 -> Conn2
91613 : Mote #2 State: Conn2 -> Conn3
93468 : Mote #2 State: Conn3 -> Oper
2731420 : Mote #2 State: Oper -> Lost
```

The “mote state” trace prints OS timestamp (ms), Mote ID, and state change for every transition. This trace shows the mote going through the joining states between **Idle** and **Operational**. After a length of time, the mote drops from the network and is shown to transition to the **Lost** state.

3.26.10 trace netmode

```
> trace netmode on

224235 : Check Steady State Start
224238 : Check Steady State. Check.
284243 : Check Steady State. Check.
344244 : Check Steady State. Check.
404245 : Check Steady State. Check.
464247 : Check Steady State. Check.
524248 : Check Steady State. Check.
584249 : Check Steady State. Check.
644250 : Check Steady State. Check.
704251 : Check Steady State. Check.
764253 : Check Steady State. Check.
824254 : Check Steady State. Check.
884255 : Check Steady State. Check.
884256 : Check Steady State. Set Steady State.
884257 : Check Steady State Stop
```

This trace shows the manager checking to make sure that at least ten minutes has elapsed since the last mote has joined. The manager will check every minute, and after ten minutes have elapsed it will transition the network from **Building** to **Steady State**.

3.26.11 trace opt

```
> trace opt on  
  
884258 : Optimization Start  
884262 : Optimization Delete  
884272 : TPLGBLD_OPTDEL for #2
```

This trace prints out all steps of the topology optimization process. In this example, an optimization delete cycle occurs for the only mote in the network.

3.26.12 trace power

```
> trace power on  
  
550296 : Power Cost Calc. Try Add 1 RX-links. Max Power 65534  
550316 : Pwer Cost Calc: TxCost 110, RxCost 65, NumTx: 4, NumRx 4, PowerCost 348
```

This trace shows the calculations the manager makes when trying to add new links to a mote. Each mote reports a maximum allowed current *maxStCur* during the joining process. If a new link would push the total power above *maxStCur*, the manager does not add this link.

3.26.13 trace rawio_enc (reserved)

```
> trace rawio_enc on

171656 : SecRX len=52
 000 : 27 32 32 1e 40 02 7e 5c 10 01 00 02 00 17 9e 95 2b 84 e5 fe
 020 : 00 7e 77 f7 99 00 00 37 eb f7 00 e0 20 ba d6 e8 29 87 67 1e
 040 : 27 1a ad 64 36 c3 4c 2e af ce fa 6a
```

This trace prints all the raw bytes of packets as they arrive at or are sent by the manager, in encrypted format. The packets are printed with 20 bytes per line.

3.26.14 trace route

```
> trace route on  
1083138 : ROUTE. New #5 Hops 3. 3-5-0
```

This trace shows the manager calculating downstream source routes for motes. The source route in this example is to mote 5 and goes AP-3-5. There is always a zero appended at the end to show the end of a route.

3.26.15 trace spl_task & trace spl_ack (reserved)

```
> trace spl_task on
> trace spl_ack on

3138150 : SPL_TASK: RX  type=22,REQ RLBL , seqNo=1 retNo=0
3138152 : SPL_TASK: Packet received. Len=8
3138153 : SPL_TASK: Wait Packet
3138154 : SPL_TASK: Packet Type 22
3138161 : SPL_TASK: Command Type 22 Result 0
3138163 : SPL_TASK: TX  type=22,ACK RLBL , seqNo=1 retNo=0
3138177 : SPL_TASK: apinotif_data Res 0
3138179 : SPL_TASK: TX  type=20,REQ RLBL , seqNo=1 retNo=0
3138204 : SPL_TASK: RX  type=20,ACK RLBL , seqNo=1 retNo=0
3138205 : SPL_TASK: Packet received. Len=5
3138206 : SPL_TASK: Acknowledge timeout 27 msec
3138209 : 3138221 : SPL_TASK: apinotif_data Res 0
SPL_TASK: Acknowledge Received
```

This trace shows the serial protocol (spl) interactions between the manager and an external application. spl_task trace shows downstream (TX) messages, and spl_ack trace show upstream (RX) responses. The example above shows the first packets exchanged between the manager and the Stargazer application.

3.26.16 trace stats

```
> trace stats on  
  
1383718 : STAT: #5: ASN Rx/Tx = 59692/59686, latency = 43, hops = 2  
1383719 : STAT: new average hops10=20 latency/10=23
```

When this trace is active, all packets arriving at the manager have their hops and latency printed.

- OS Timestamp (ms)
- ASN Rx/Tx: When the packet was received (Rx) by the manager, and when the packet was generated (Tx) by the mote or received by the mote on the *sendTo* API.
- latency (ms): # of slots (delta between Rx/Tx ASN) * 7.25 ms slot length.
- hops: how many hops this packet took
- new average: The averages for hops and latency are printed on the next line. Hops10 is in units of 0.1 hops and latency/10 is in units of 10 ms, so the averages printed in this example represent 2.0 hops and 230 ms.

3.26.17 trace timeout

```
>trace timeout on  
224238 : Reliable Timeout Mote: #2 Pkt:Act RetrNum: 1 Timeout 180
```

This “reliable timeout” trace will show all reliable downstream packets


- OS Timestamp (ms)
- Mote: To whom the packet was sent
- Pkt: Packet type (here an “Activate” packet)
- RetrNum: Retry number (here the first try)
- Timeout: Timeout in seconds before the next retry

3.26.18 trace tplslock (reserved)

```
> trace tplslock on  
  
1261676 : tpls_shortLock()  
1261678 : tpls_shortUnlock()  
1262023 : tpls_shortLock()  
1262026 : tpls_shortUnlock()  
1262843 : tpls_shortLock()
```

This trace shows when the manager locks and unlocks the topology for the network. In between these events, no changes to routes or bandwidth can be made. These events are frequent, and no reason for the lock is shown, so this is normally only used in manager testing during internal development.

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 Linear Technology 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 Linear Technology.

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.

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

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

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

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

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

© Linear Technology Corp. 2012-2016 All Rights Reserved.