

Dust Networks

SmartMesh[®] IA-510
Manager XML API Guide
Industrial



Trademarks

SmartMesh-XR, SmartMesh-XT, and SmartMesh IA-510 are trademarks of Dust Networks, Inc. The Dust Networks logo, Dust, Dust Networks, and SmartMesh are registered trademarks of Dust Networks, Inc. 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. 2007, 2008, 2009, 2010. All Rights Reserved

Document Number: 040-0069 rev 3 SmartMesh IA-510 (I) Manager XML API Guide
Last Revised: October 26, 2010

Contents

About This Guide

Related Documents	v
Conventions Used in This Guide	v
Revision History	vi

1 Introduction

SmartMesh Manager	2
Control and Notification Channels	2
Control Channel.....	2
Notification Channel	3
Concurrent Client Support.....	3
Protocols	3
XML	3
XML-RPC.....	4
SmartMesh Manager XML API	5
Configuration Schema	5
Notification Schema	6
SmartMesh Manager XML API Commands.....	6

2 Using the SmartMesh Manager XML API

Network Activities	10
Network Message Formats	10
Configuration Messages	10
Notification Messages	11
XML Requirements	11
Using the Control Channel	12
Logging In to the Control Channel	12
Logging Out of the Control Channel	13
Requesting Configuration Values	14

Requesting All Children of a Configuration Element .	14
Specifying a Configuration Element's Own Level	15
Specifying Multiple Child Elements	17
Requesting the Network Topology	18
Requesting Network Statistics	19
Setting a Configuration Object	19
Changing a Configuration Object	20
Deleting a Configuration Object	21
Sending Commands to the Manager	22
Receiving Data Over the Notification Channel	23
Subscribing to the Notification Channel	23
Mote Joining	25
Network Bandwidth Control	25
End-to-end Wireless Communication	28
Reliable Wireless Communication	28
Best-effort Wireless Communications	28
Network ID	29
Network Security	29
Common Join Key Mode	29
ACL Mode	30

3 SmartMesh Manager XML API Reference

Network Commands	31
activateAdvertising	33
activateFastPipe	33
cancelOtap	35
cli	35
createConfig	36
deactivateFastPipe	37
decommissionDevice	37
deleteConfig	38
exchangeJoinKey	39
exchangeMoteJoinKey	40
exchangeNetworkId	42
exchangeNetworkKey	43
exchangeSessionKey	44
getConfig	45
getLatency	46
getTime	47
login	47
logout	48

pingMote	48
reset <object>	49
sendRequest	50
sendResponse	52
setConfig	53
startOtap <numRetries>	54
subscribe	55
unsubscribe	56
Error Messages	57
Configuration Schema Reference	58
element config	59
Elements Used in Configuration Commands	60
element config/System	61
element config/Network	66
element config/Users	85
element config/Security	86
element config/Motes	89
element config/Paths	103
element config/Alarms	108
element config/EventLog	110
Notification Schema Reference	111
element notifications	111
element notifications/data	112
element notifications/event	114
element notifications/log	140
complexType alarmInfo	141
element notifications/cli	142

Index

About This Guide

This guide provides support for developers building client applications or scripts that interact with a SmartMesh-enabled network through the SmartMesh® manager.

This guide contains the following chapters:

Chapter 1, Introduction, includes a description of control and notification channels, role of the manager between the notes and the user views of the data, and suggested uses for the XML API.

Chapter 2, Using the SmartMesh Manager XML API, includes general guidance on how to use the XML API (prerequisite steps, scripting tips and examples, basic tasks such as login and subscribe).

Chapter 3, SmartMesh Manager XML API Reference, is a reference section that fully documents the schemas with descriptions, notes, and examples.

Related Documents

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


- *SmartMesh Manager XML API Guide* (this guide)
- *SmartMesh Manager Serial API Guide*


Conventions Used in This Guide

The following conventions are used in this guide:

- `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.

 **Note:** Notes provide more detailed information about concepts.

 **Caution:** Cautions advise you about actions that might result in a loss of data.

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

Revision History

Revision	Date	Description
040-0069 rev 1	6/12/2009	Final (product release).
040-0069 rev 2	11/9/2009	
040-0069 rev 3	8/20/2010	

Introduction

This chapter introduces the SmartMesh manager XML application programming interface (API) and the SmartMesh-enabled network.

A SmartMesh-enabled network contains:

- A network of motes (remote sensors) that sense data and exchange information packets via radio.
- A manager that configures the network and the motes, receives data packets from the motes, and streams data to a client application.
- Clients that give the manager configuration instructions and receive mote and network data from the manager in XML format.

The SmartMesh manager XML API is an Extensible Markup Language (XML) interface that lets a client application send requests to the manager and receive responses and other data from the manager via XML-RPC.

Note: Note that the *IA-510 Manager XML API Guide* can be used with both the D2511 manager and the PM2511 embedded manager. Figure 1 illustrates how a D2511 manager interfaces to the network and the associated client. For applications that use the PM2511 manager, the diagram would show the manager as an embedded part of the OEM Controller.

Clients exchange XML messages with the manager, which in turn communicates with the motes in the network.

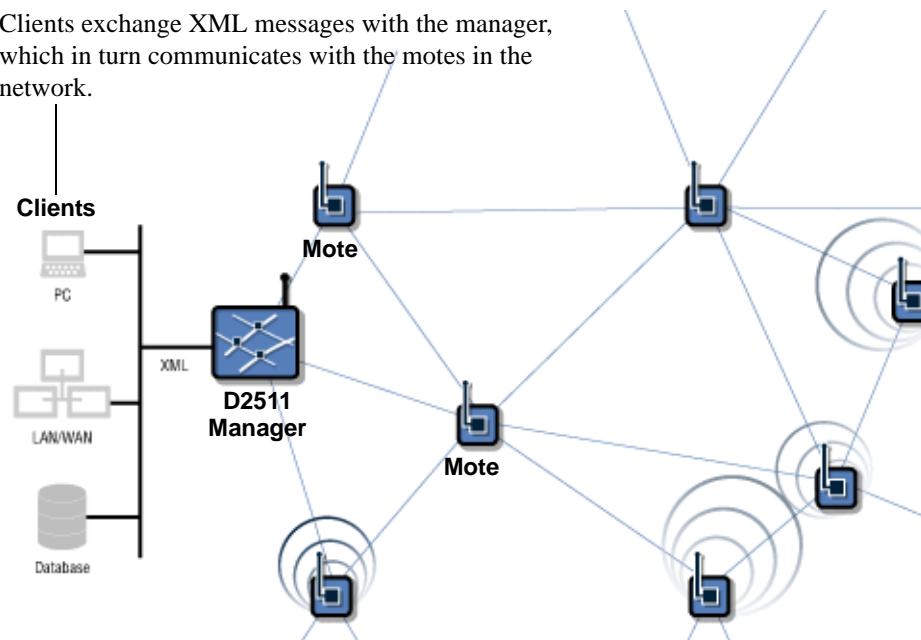


Figure 1 A SmartMesh-enabled Network

SmartMesh Manager

SmartMesh manager is the bridge between a client application and a SmartMesh-enabled network. It functions as the server that provides the XML-RPC interface between client applications and the remote wireless network of motes.

The manager exchanges packets with motes in the network. The packets, which may be passed from mote to mote en route to their destination, contain data and network control information.

The manager uses two channels for communication with a client: a two-way *control channel* that permits requests and responses between the manager and a client, and a one-way *notification channel* that streams data from the manager to a client.

Control and Notification Channels

The notification channel and control channel are two ports over which a client establishes a connection with the manager's IP address.

The control channel is used to transmit commands from the client application to the manager, and receive responses generated from the manager.

The manager uses the notification channel to transmit asynchronous notifications to the client. The client application uses a subscribe command on the control channel to request a notification channel.

Control Channel


The manager and client applications use the control channel to exchange commands and information about a SmartMesh-enabled network. The first command from the client to the manager is a login, and the manager returns a token that the client uses in all subsequent commands.

All communications over the control channel use the format described by the XML-RPC API, and take the form of HTTP requests.

By default, the control channel port is pre-configured as 4445.

Notification Channel

The manager uses the notification channel to stream data and network events to the client. To subscribe to the notification channel, clients use a subscribe method call over the control channel, and the manager returns a token and the port number at which the notification channel is available. Clients then initiate a TCP connection to the manager on the notification port, and send an authorization request containing the token to the manager over the TCP connection. As long as the connection remains open, the manager returns a stream of XML sensor data and network events.

 **Note:** Since the notification channel is used for streaming data, it uses XML messages over a TCP/IP connection. *The notification channel does not use XML-RPC.*

Client scripts or applications listen on the notification channel and process the XML-wrapped data.

Communications over the notification channel use the format described by the notification XML schema. See “[Notification Schema](#)” for a general overview of the notification schema’s contents, and see “[Notification Schema Reference](#)” for a reference guide.

Concurrent Client Support

The SmartMesh manager can support up to 5 live simultaneous connections to client applications, such as applications for monitoring and controlling the network, and data-listening applications. Each active application has its own login token that identifies it to the manager.

Protocols

SmartMesh-enabled network client applications use XML and XML-RPC to communicate with the manager.

XML

XML is a self-describing, human-readable, extensible metalanguage that uses descriptive tags to define the organizational structures and items in a data set or document. The SmartMesh manager XML API defines tags that describe the components and configuration of SmartMesh-enabled networks.

For example, the following XML stanza encapsulates network data about a live SmartMesh-enabled network:


```
<config>
  <Network>
    <netName>myNet</netName>
    <networkId>1</networkId>
    <optimizationEnable>true</optimizationEnable>
  </Network>
</config>
```

```
<numMotes>100</numMotes>
<maxMotes>200</maxMotes>
</Network>
</config>
```

XML-RPC

XML-RPC is a simple protocol that enables a computer to execute remote procedures on another computer using HTTP and XML. In a typical XML-RPC exchange, a client computer uses HTTP to send an XML document containing a method name and arguments to a server. The server invokes the method with the arguments, and then wraps up the return value of the method in another XML document, which it sends back to the client.

The SmartMesh manager implements a number of methods for querying and setting configuration of the manager and the network. Most of these methods interact with a configuration document on the manager. This configuration document is defined by the SmartMesh manager XML API schemas. See “[Network Commands](#)”.

 **Note:** The XML-RPC server on the manager uses chunked HTTP transfers for large amounts of data.

For more information about XML-RPC, see the following Web sites:

- <http://www.xmlrpc.com/> (specification, information, and examples)
- <http://ws.apache.org/xmlrpc/index.html> (Apache implementation)
- <http://www.dom4j.org/index.html> (Java open-source XML framework)
- http://xmlrpc-c.sourceforge.net/examples/synch_client.c. (C examples)
- <http://www.jmarshall.com/easy/http/> (chunked HTTP transfers)

SmartMesh Manager XML API

The SmartMesh manager XML API is a set of XML-RPC functions and a suite of XML schemas that describe the formats for all exchanges between clients and the manager. The schemas document the elements in the configuration statements exchanged on the control channel, and the data streamed over the notification channel.

Configuration Schema

The configuration schema is part of the interface that gives client applications programmatic control of a SmartMesh-enabled network. Use the SmartMesh manager XML API commands to perform operations on the elements in the configuration schema (see “[SmartMesh Manager XML API Commands](#)”).

The table below briefly describes the configuration schema’s major components and provides links to detailed reference information.

System	Identifies basic system properties such as control and notification ports and system location, name, and time. See “ element config/System. ”
Network	Configures network security and performance options for a SmartMesh-enabled network. See “ element config/Network. ”
Users	Specifies a user's name, password, and privileges. See “ element config/Users. ”
Security	Configures the network security mode and specifies the list of devices that may be accepted into the network. See “ element config/Security. ”
Motes	Contains individual mote settings and mote statistics. See “ element config/Motes. ”
Paths	Contains settings and statistics for communication paths between motes. See “ element config/Paths. ”
Alarms	Contains a list of currently active alarms. See “ element config/Alarms. ”
EventLog	Logs system and network events. See “ element config/EventLog. ”

Notification Schema

The notification schema is an XML interface that describes the format for all data and events that the manager routes from motes to client applications. The following table lists notification types that may be present on the notification channel. See [“Notification Schema Reference”](#).

data	Notifications containing data originated by the sensor.
event	Notifications about network and system events.
log	Notifications about important events logged by the manager.
cli	Output of CLI commands invoked remotely.

SmartMesh Manager XML API Commands

Client applications perform operations on a SmartMesh-enabled network by calling methods remotely on the SmartMesh manager. Valid methods include the following operations.

Table 2 Network Command Summary

Network Command	Description
activateAdvertising	Turns on advertising.
activateFastPipe	Turns on a high bandwidth channel between the manager and a mote.
cancelOtap	Cancels an OTAP in progress.
cli	Tunnels a given CLI command through to manager's CLI.
createConfig	Creates a configuration object.
deactivateFastPipe	Turns off a pipe between the manager and a mote.
decommissionDevice	Initiates mote decommissioning.
deleteConfig	Deletes a configuration object.
exchangeJoinKey	Initiates an update of the join key on the manager and all connected motes in the network using the common join key.
exchangeMoteJoinKey	Initiates an update of the join key on a specified mote.
exchangeNetworkId	Updates the network ID and initiates its distribution to the motes.
exchangeNetworkKey	Generates a new network key and initiates its distribution to the motes.
exchangeSessionKey	Generates a new session key and initiates its distribution to the mote(s).
getConfig	Requests configuration information.
getLatency	Requests estimate of data latency for a specified mote.

Table 2 Network Command Summary

Network Command	Description
getTime	Requests the current time.
login	Logs in to the control channel.
logout	Logs out of the control channel.
pingMote	Sends a packet to a mote requesting a response.
reset <object>	Resets an object, such as a mote.
sendRequest	Sends a request packet to a mote.
sendResponse	Sends a response packet to a mote.
setConfig	Sets configuration values.
startOtap <numRetries>	Begins the automatic process of upgrading mote software using over-the-air programming (OTAP).
subscribe	Subscribes to the data notification channel.
unsubscribe	Shuts down a notification TCP connection to a client.

These commands take specified parameters that often determine the action that the command invokes. For example, the reset command can be called on a system, a network, a mote, a statistic, or an event log. The following sample stanza is what a client application would send the manager to reset mote 23:

```
<methodCall>
  <methodName>reset</methodName>
  <params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>mote</string></value>
    </param>
    <param>
      <value><int>23</int></value>
    </param>
  </params>
</methodCall>
```

For full documentation and examples of the SmartMesh manager API commands, see [“Network Commands”](#).

Using the SmartMesh Manager XML API

This chapter explains how client applications use the SmartMesh XML APIs to program and monitor a SmartMesh-enabled network.

SmartMesh manager XML API operations can be scripted or incorporated programmatically into client applications.

A script can be complex, or a simple sequence of requests, such as:

1. Login to the control channel.
2. Subscribe to the notification channel.

Scripts can be written in a variety of programming languages, such as Perl, C, and Java.

An application that uses the manager API can provide a graphical interface to the manager API and enable users to interact with sensor networks.

This chapter discusses the following topics:

- [Network Activities](#)—High-level view of a client's interactions with a network's manager.
- [Network Message Formats](#)—The packaging requirements for messages passed to the SmartMesh manager.
- [Using the Control Channel](#)—Examples of basic requests and responses between a client application and the SmartMesh manager.
- [Receiving Data Over the Notification Channel](#)—Processing the stream of data from the network.

Network Activities

This section lists the basic procedures a client application uses to interact with a SmartMesh manager.

- Establish a connection to the manager’s control channel. See [“Logging In to the Control Channel.”](#)
- Subscribe to the manager’s notification channel. See [“Subscribing to the Notification Channel.”](#)
- Request configuration information. See [“Requesting Configuration Values.”](#)
- Send network commands. See [“Sending Commands to the Manager.”](#)
- Log out of the control channel. See [“Logging Out of the Control Channel.”](#)

Network Message Formats

Network messages are defined in the SmartMesh XML schemas and conform to XML requirements and the XML-RPC application.

Configuration Messages

The format for control channel messages is XML-RPC over HTTP. Control channel messages include:

- Commands sent from a client to the manager over the control channel
- Responses sent from the manager to a client over the control channel

Both of these types of messages are sent in an HTTP wrapper. The following is an example of an HTTP wrapper for an XML-RPC method call that a client sends to the manager:

```
Host: 127.0.0.3
User-Agent: RPC::XML::Client/1.20 libwww-perl/5.79
Content-Length: 80
Content-Type: text/xml
```

Examples in the following sections assume the HTTP wrapper, instead of including it in each message.

Notification Messages

The format for messages sent over the notification channel is XML over TCP/IP. Notification channel messages include:

- Authorization requests and responses exchanged over the notification channel between a client and the manager
- Asynchronous notifications sent over a data channel from the manager to a client

A notification session is initiated by a client making a subscribe call on the control channel. A client then connects to a TCP port on the manager and the manager pushes events asynchronously to the client.

Messages over the notification channel use the formats defined in the SmartMesh notification schema. See [“Notification Schema Reference” on page 111](#).

XML Requirements

Client applications specify child elements within a request element’s opening and closing tags. If any of the requested element’s children themselves contain child elements, the opening tag for the parent must occur before any of its child elements, and the parent’s closing tag must occur before the opening tag for another tag element at its hierarchy level.

SmartMesh messages conform to the XML convention that an element’s name indicates the kind of information it contains. For example, the name of the SmartMesh `<numJoins>` tag element indicates that it specifies the number of times the mote joined the network.

When tagging items in an XML-compliant document or data set, always enclose the item in paired opening and closing tags. XML is stricter in this respect than HTML, which sometimes uses only opening tags. The following example shows paired opening and closing tags:

```
<numJoins>24</numJoins>
```

If an element has no value assigned, you can represent it either as a pair of opening and closing tags with nothing between them or as a single tag with a forward slash after the tag name. For example, `<systemName/>` represents the same empty tag as `<systemName></systemName>`.

Using the Control Channel

This section provides examples of a client application invoking operations on a SmartMesh manager using the SmartMesh manager XML API.

Each operation involves sending an XML-RPC request over the manager's control channel and receiving a response.

- [Logging In to the Control Channel](#)
- [Subscribing to the Notification Channel](#)
- [Logging Out of the Control Channel](#)
- [Requesting Configuration Values](#)
- [Requesting the Network Topology](#)
- [Setting a Configuration Object](#)
- [Changing a Configuration Object](#)
- [Deleting a Configuration Object](#)
- [Sending Commands to the Manager](#)
- [Requesting Network Statistics](#)

Logging In to the Control Channel

To begin receiving data from a SmartMesh-enabled network, the client application establishes contact with the manager by logging in to the manager's control channel, and then subscribes to the notification channel.

If you simply want to log in and begin receiving data, use the instructions in this section to log in to the control channel and then skip ahead to "[Subscribing to the Notification Channel](#)" on page 23.

Use this procedure to log in to the control channel:

1. Establish a TCP/IP connection to the control channel port (the default is 4445) at the manager's IP address.
2. Log in to the control channel by sending the manager an XML request within an HTTP wrapper containing the login method call with two parameters: username and password, as shown in the example below.

In response to the login request, the manager returns an authorization token. Use this token as the first parameter in all subsequent calls to the manager.

Your client application sends this login request in the body of the HTTP request:

```
<methodCall>
  <methodName>login</methodName><params>
    <param>
      <value><string>admin</string></value>
    </param>
    <param>
      <value><string>admin</string></value>
    </param></params>
</methodCall>
```

The manager returns this response in the body of the HTTP response:

```
<methodResponse>
  <params><param>
    <value><string>dn7c049dc1-3</string></value>
  </param></params>
</methodResponse>
```

The manager activates the token when it sends it to the client application, and expects the token as the first parameter in all subsequent control channel requests from the client. The manager has a limit of 100 active tokens on the control channel.

Logging Out of the Control Channel

Log out of the control channel by sending the logout method to the manager with a single parameter: the token that the manager returned in response to your login request.

The manager responds by cancelling the token and de-allocating resources for the login session.

Your client application sends this request:

```
<methodCall>
  <methodName>logout</methodName>
  <params><param>
    <value><string>dn7c049dc1-3</string></value>
  </param></params>
</methodCall>
```

The manager returns this response:

```
<methodResponse>
  <params>
    <param><value>
      <string>OK</string>
    </value></param>
  </params>
</methodResponse>
```

Logging out of the control channel does not cancel your subscription to the data stream coming over the notification channel.

Requesting Configuration Values

Request the values for a configuration element by requesting its parent object. The response will include values for children, and the children of any children, down to the hierarchy depth specified as the first parameter in a getConfig call.


Specifying a depth of “all” as the first parameter in a getConfig call requests the values for all of an element’s children.

Specifying a depth of “0” requests only the immediate values of the element itself, and is generally used when requesting a single element, such as a motelId.

Specifying a depth of “1” requests values for the element’s immediate children. This depth is generally used either to collect lists of elements or to collect all the values of a single element.

Depth can be set at any number greater than zero.

The three request and response examples in this section request configuration information for the Network object including all of its sublevels, none of its sublevels, and one of its sublevels, respectively.


 **Important:** The third parameter of the getConfig call is an XML fragment. In order to fit into the XML-RPC call, the XML tags must be escaped. This is not shown in the examples below.

Requesting All Children of a Configuration Element

To request all sublevels of the Network object, your client application sends this request:

```
<methodCall>
  <methodName>getConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>all</string></value>
    </param>
    <param>
      <value><string>
        <config>
          <Network/>
        </config>
      </string></value>
    </param></params>
</methodCall>
```

The manager returns this response, which contains all sublevels except Statistics.

 **Note:** Statistics subobjects (under Network, Mote, and Path) must be queried separately. This is because Statistics contains a very large amount of data. For performance reasons, this data is only returned if the client specifically requests it. See [“Requesting Network Statistics.”](#)

```
<methodResponse>
  <params><param>
    <value><string>
      <config>
        <Network>
          <netName>myNet</netName>
          <networkId>1</networkId>
          <optimizationEnable>true</optimizationEnable>
          <maxMotes>100</maxMotes>
          <numMotes>10</numMotes>
          <gatewayPA>false</gatewayPA>
          <ccaEnabled>false</ccaEnabled>
          <netQueueSize>10</netQueueSize>
          <userQueueSize>10</userQueueSize>
        </Network>
        <ChannelBlackList>
        </ChannelBlackList>
        <Sla>
          <minNetReliability>95</minNetReliability>
          <maxNetLatency>20000</maxNetLatency>
          <minNetPathStability>50</minNetPathStability>
        </Sla>
      </config>
    </string></value>
  </param></params>
</methodResponse>
```

Specifying a Configuration Element’s Own Level

To request configuration values set at the element’s hierarchy level, but not any of its children, your client application sends this request:

```
<methodCall>
  <methodName>getConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>1</string></value>
    </param>
    <param>
      <value><string>
        <config>
          <Network/>
        </config>
      </string>
    </param>
  </params>
</methodCall>
```

```
</config>
</string></value>
</param></params>
</methodCall>
```

The manager responds with the values configured at the top hierarchy level in the mote profile, and the values of its immediate children:

```
<methodResponse>
  <params><param>
    <value><string>
      <config>
        <Network>
          <netName>myNet</netName>
          <networkId>1</networkId>
          <optimizationEnable>true</optimizationEnable>
          <maxMotes>100</maxMotes>
          <numMotes>10</numMotes>
          <gatewayPA>false</gatewayPA>
          <ccaEnabled>false</ccaEnabled>
          <netQueueSize>10</netQueueSize>
          <userQueueSize>10</userQueueSize>
        </Network>
        <ChannelBlackList/>
        <Sla/>
      </config>
    </string></value>
  </param></params>
</methodResponse>
```

Specifying Multiple Child Elements

To request values for two sublevels of the network, the client application requests:

```
<methodCall>
  <methodName>getConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>2</string></value>
    </param>
    <param>
      <value><string>
        <config>
          <Network/>
        </config>
      </string></value>
    </param></params>
</methodCall>
```

The manager responds with the values for the network and its one immediate sublevel. Notice that in this example, the same values are returned as if the depth was specified as “all.”

```
<methodResponse>
  <params><param>
    <value><string>
      <config>
        <Network>
          <netName>myNet</netName>
          <networkId>1</networkId>
          <optimizationEnable>true</optimizationEnable>
          <maxMotes>100</maxMotes>
          <numMotes>10</numMotes>
          <gatewayPA>false</gatewayPA>
          <ccaEnabled>false</ccaEnabled>
          <netQueueSize>10</netQueueSize>
          <userQueueSize>10</userQueueSize>
        </Network>
        <ChannelBlackList>
        </ChannelBlackList>
        <Sla>
          <minNetReliability>95</minNetReliability>
          <maxNetLatency>20000</maxNetLatency>
          <minNetPathStability>50</minNetPathStability>
        </Sla>
      </config>
    </string></value>
  </param></params>
</methodResponse>
```

Requesting the Network Topology

The following example shows how the client application requests the network topology of motes and the active radio connection paths among them. Setting a query to a depth of “1” and querying the paths element returns a list of paths. Setting a query to a depth of “1” and querying path 18 returns the values of path 18’s immediate children, thus displaying the mote IDs for path 18’s mote A and mote B.

```
<methodCall>
  <methodName>getConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>2</string></value>
    </param>
    <param>
      <value><string>
        <config>
          <Paths/>
        </config>
      </string></value>
    </param></params>
</methodCall>
```

The manager responds with the values for the path, which describe the network topology:

```
<methodResponse>
  <params><param>
    <value><string>
      <config>
        <Paths>
          <Path>
            <pathId>19</pathId>
            <moteAMac>12</moteAMac>
            <moteBMac>4</moteBMac>
          </Path>
          <Path>
            <pathId>19</pathId>
            <moteAMac>4</moteAMac>
            <moteBMac>1</moteBMac>
          </Path>
          <Path>
            <pathId>19</pathId>
            <moteAMac>5</moteAMac>
            <moteBMac>4</moteBMac>
          </Path>
        </Paths>
      </config>
    </string></value>
  </param></params>
</methodResponse>
```

Requesting Network Statistics

This section presents an example of a request for current statistics on network efficiency.

Network statistics have to be queried specifically, unlike other SmartMesh getConfig queries. That is, the client must specify all the way down to the hierarchical level of the statistical element it is requesting. This is because the manager does not automatically return all of the children of requested statistical elements.

```
<methodCall>
  <methodName>getConfig</methodName>
  <params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>1</string></value>
    </param>
    <value><string>
      <config><Network>
        <Statistics>
          <statCur/>
        </Statistics>
      </Network></config>
    </string></value>
  </params>
</methodCall>
```

The manager's response will contain:

```
<config>
  <Network>
    <Statistics>
      <statCur>
        <netReliability>100</netReliability>
        <netPathStability>98.182</netPathStability>
        <netLatency>7391</netLatency>
      </statCur>
    </Statistics>
  </Network>
</config>
```

Setting a Configuration Object

To add a configuration object such as a user, use the *createConfig* method call and include the tag elements that represent the entire hierarchy down to the level of the configuration element you want to add.

! **Warning!** Make sure the object identifiers for your new object are unique, to avoid changing an existing element instead of adding a new one.

When changing a configuration object, the manager validates the new configuration before accepting it. If the validation fails, the manager returns an error string containing a description of the error.

For example, to add a user named “Alice” and specify its properties, the client requests:

```
<methodCall>
  <methodName>createConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>/config/Users/User
      </string></value>
    </param>
    <param>
      <value><string>
        <config>
          <Users>
            <User>
              <userName>Alice</userName>
              <password>guie4sdc</password>
              <privilege>superuser</privilege>
            </User>
          </Users>
        </config>
      </string></value>
    </param></params>
  </methodCall>
```

If a name string is too long, the manager returns the following string:

```
<methodResponse>
  <fault>
    <value> <string> error: name validation: string length out
    of range. Length should be between 1 and 16.
    </string> </value>
  </fault>
</methodResponse>
```

Changing a Configuration Object

To change a configuration object, use the setConfig method call and include the tag elements that represent the entire hierarchy down to the configuration elements you want to change.

To change the network name to “rootNet” the client application requests:

```
<methodCall>
  <methodName>setConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>
        <config>
          <Network>
```

```

        <netName>roofNet</netName>
    </Network>
</config>
</string></value>
</param></params>
</methodCall>

```

The manager returns the object:

```

<methodResponse>
  <params><param>
    <value><string>
      <config>
        <Network>
          <netName>roofNet</netName>
          <networkId>1</networkId>
          <optimizationEnable>true</optimizationEnable>
          <maxMotes>100</maxMotes>
          <numMotes>10</numMotes>
          <gatewayPA>false</gatewayPA>
          <ccaEnabled>false</ccaEnabled>
          <netQueueSize>10</netQueueSize>
          <userQueueSize>10</userQueueSize>
        </Network>
      </ChannelBlackList>
    </ChannelBlackList>
    <Sla>
      <minNetReliability>95</minNetReliability>
      <maxNetLatency>20000</maxNetLatency>
      <minNetPathStability>50</minNetPathStability>
    </Sla>
  </Network>
</config>
</string></value>
</param></params>
</methodResponse>

```

Deleting a Configuration Object

To delete a configuration object, use the deleteConfig method call and include the tag elements that represent the entire hierarchy down to the configuration element you want to delete.

The first example deletes the entire ACL list. The second example deletes MAC address 00-00-00-00-00-00-00-00 from the ACL list.

To delete the entire ACL list, the client application requests:

```

<methodCall>
  <methodName>deleteConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>

```

```

    <config>
      <Security>
        <Acl/>
      </Security>
    </config>
  </string></value>
</param></params>
</methodCall>

```

The manager sends “OK” if the delete is successful or a validation error if it is not successful.

To delete only MAC address 00-00-00-00-00-00-00-00 from the ACL list, the client application requests:

```

<methodCall>
  <methodName>deleteConfig</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>
        <config>
          <Security>
            <Acl>
              <Device>
                <macAddress>00-00-00-00-00-00-00-00</macAddress>
              </Device>
            </Acl>
          </Security>
        </config>
      </string></value>
    </param></params>
  </methodCall>

```

And the manager responds with “OK”:

```

<methodResponse>
  <params><param>
    <value><string>OK</string></value>
  </param></params>
</methodResponse>

```

Sending Commands to the Manager

The commands that the client application sends to the manager include commands to reset an object, exchange security keys, and send an opaque data packets to a mote.

For detailed reference information about network commands, see [“Network Commands” on page 31](#).

Reset commands take the following format:

```

<methodCall>
  <methodName>reset</methodName>

```

```
<params>
  <param>
    <value><string>network</string></value>
  </param>
</params>
</methodCall>
```

Receiving Data Over the Notification Channel

This section describes how to subscribe to the notification channel, and gives examples of the formatting of data sent to a client over the notification channel.

Subscribing to the Notification Channel

The notification channel is a data channel that the manager uses to publish network and sensor data to a port on your computer.

To subscribe to the notification channel:

1. Send the token to the notification channel port with a parameter value for the events to which you want to subscribe. To subscribe to all events, use, “all.”
2. Send the subscribe method call to the manager, which responds by returning:
 - token
 - The port number of the notification channel

Your client application sends a subscribe request:

```
<methodCall>
  <methodName>subscribe</methodName><params>
    <param>
      <value><string>dn7c049dc1-3</string></value>
    </param>
    <param>
      <value><string>all</string></value>
    </param></params>
</methodCall>
```

The manager returns this response:

```
<methodResponse>
  <params><param>
    <value><array><data>
      <value><string>dn7c049dc1-3</string></value>
      <value><int>port</int></value>
    </data></array></value>
  </param></params>
</methodResponse>
```

When a manager receives a subscribe request, it starts listening on the specified port for incoming TCP/IP connections, expecting an authorization request.

The client sends the token to the port identified in the manager's response using the following format:

```
<dustnet version="1.0">
<authrq>
  <token>dn7c049dc1-3</token>
</authrq>
</dustnet>
```

The manager accepts or rejects the token. A rejection takes the form:

```
<dustnet version="1.0">
<authrs>
  <error>invalid authorization</error>
</authrs>
</dustnet>
```

On acceptance, a notification channel authorization begins the XML stream that carries all of the messages that the manager sends over the notification channel. A sample notification channel authorization:

```
<dustnet version="1.0">
<authrs/>
```

Note that the `dustnet` element does not end. All of the data and event messages are contained in the `dustnet` element.

Following acceptance, messages about data and events sent over the notification channel are enclosed in descriptive tags:

```
<event>
  <timeStamp>1094853387499</timeStamp>
  <eventId>100</eventId>
  <sysBootUp/>
</event>

<data>
  <moteId>19</moteId>
  <macAddr>00-11-00-11-00-11-00-22</macAddr>
  <time>1094853397499</time>
  <payload type="80">A9FD64E12C</payload>
</data>
```

When a client application closes the notification channel, the manager unsubscribes the client.

Network Bandwidth Control

IA-510 network managers support dynamic bandwidth to accommodate the bandwidth requirements of complex applications common in monitoring and control. For example, an IA-510 network can support request/response maintenance traffic from a controller or gateway down to a network device while simultaneously activating a fast pipe to transfer a large amount of data at the request of a worker in the field. The IA-510 network manager precisely allocates network resources to support the bandwidth requirements of the application while maintaining ultra-low power consumption across the network. To support these applications, bandwidth can be requested by either the network manager using the manager API commands or requested by network devices through bandwidth service requests. In response to these bandwidth controls, the network manager will schedule links in the network to increase bandwidth, or in some cases may activate a fast bandwidth pipe.

This section describes the IA-510 manager API commands for both requesting bandwidth and controlling the bandwidth allocated to service requests. The IA-510 network manager API enables the following:

- Limits on Service-Requested bandwidth—define maximum bandwidth that may be granted to service requests.
- Manager-requested bandwidth—defines network-wide minimum bandwidth.
- Allocated bandwidth—return the total bandwidth allocated to a specific device.
- Pipe activation—enables the fast pipe to be activated or deactivated through the manager API and provides visibility to the pipe status.

Since fast pipes may be activated by the network manager as a result of a manager API command or from a service request from a network device, it is worth noting that the device that requested the bandwidth owns and controls it. That is, a pipe activated by service request can only be deactivated by a subsequent request from the same device. Similarly, a pipe activated by manager API can only be deactivated by manager API.

In addition to manager-initiated bandwidth control, there are also mechanisms by which the mote can exercise bandwidth control. For more information, see the “Bandwidth Services” section in the *IA-510 Mote Serial API Guide*.

Table 3 Manager API Bandwidth Commands and Parameters

Manager API Parameter	Description
User-Settable Bandwidth Limits	
config/Network/minServicesPkPeriod	Limits non-pipe services. Defines maximum bandwidth (minimum data period) that a single mote may be allocated for the total of non-pipe, user requested bandwidth. Limits service requests from mote

Table 3 Manager API Bandwidth Commands and Parameters (Continued)

Manager API Parameter	Description
config/Network/minPipePkPeriod	Limits pipe bandwidth. Limits bandwidth on all pipes (both manager-API requested and pipes as a result of service request). Most useful for regulating service requests from field devices.
Manager-API Driven Bandwidth Requests	
config/Network/requestedBasePkPeriod	Network-wide minimum bandwidth. Defines network-wide minimum bandwidth (maximum packet period). Note that the config/Network/requestedBasePkPeriod parameter applies to manager-controlled bandwidth, which is independent from bandwidth requested through mote service requests.
Allocated Bandwidth	
config/Motes/Mote/allocatedPkPeriod	Total bandwidth allocated to a specific mote. Returns the total usable non-pipe bandwidth (1/allocatedPkPeriod) that was allocated to the mote. May be compared against /config/network/minServicePkPeriod limit. Includes bandwidth allocated from: <ul style="list-style-type: none"> • Network/requestedBasePkPeriod • mote service requests
config/Motes/Mote/allocatedPipePkPeriod	Pipe bandwidth. Returns usable bandwidth allocated to the pipe.
notifications/event/netServiceDenied	XML API notifications that indicate when a network service has been denied. The minServicesPkPeriod limit has been reached or there is a bottleneck in the network, as indicated by the config/Motes/Mote/needNeighbor flag.
Pipe Activation	

Table 3 Manager API Bandwidth Commands and Parameters (Continued)

Manager API Parameter	Description
activateFastPipe command	XML API command to turn on pipe. Requests activation of a fast pipe to a specific Mote. Parameters are token, destMacAddress, and pipeDirection. Valid values for pipeDirection are UniUp (unidirectional-Up), UniDown (unidirectional-Down), and Bi (bidirectional). The values are case sensitive.
Notifications/event/netPipeOn	XML API pipe activation notification. Notification sent by the network manager via manager API, signifying a pipe has been activated. The notification includes: macAddr, allocatedPipePkPeriod. The value for allocatedPipePkPeriod will match that of config/Motes/Mote/allocatedPipePkPeriod.
config/Motes/Mote/pipeStatus	Status of pipe. Pipe states are off, pending, on_bi, on_up, and on_down

End-to-end Wireless Communication

Users may choose reliable or best-effort communication for transmissions between the client and the network device. The network manager supports reliable communication in both downstream and upstream directions. Downstream communication refers to packets sent from the client to the network device. Upstream communication refers to packets sent from the network device to the client.

Reliable Wireless Communication

In reliable communication, packets sent wirelessly are acknowledged end-to-end, providing confirmation to the application layer that the packet was successfully delivered. If the confirmation is not received, the manager will re-transmit the packet. With reliable communication, only one packet per destination address can be in transit in the network at a time. Additional packets are queued by the manager.

When sending a reliable request downstream, use the *sendRequest* command with the

isreliable field enabled. When the manager receives the response, it forwards it to the client via a */notifications/data* notification.


If the network sends a reliable upstream packet, the manager delivers it to the client using the */notifications/data* notification with the *isreliable* field included. The client should send the corresponding reply using the *sendResponse* command.

Applications requiring guaranteed delivery should use the reliable communication mechanism.

Best-effort Wireless Communications

In best-effort communication, packets sent wirelessly are forwarded to the application layer and no explicit acknowledgement packets are generated. Therefore, the application receives no feedback about the success of individual packet delivery. On the other hand, overall network power consumption is lower since there is no traffic from the acknowledgement. In addition, since more than one best-effort packet may be in transit to a specific destination at a time, network throughput may be higher. This method is best suited when no application acknowledgement is required or when a small percentage of lost packets is tolerable.

When sending a best-effort packet downstream, use the *sendRequest* command with the *isreliable* field disabled.

 **Important!** Broadcast packets (packets sent to all motes in the network) must always be sent using best-effort communication.

Network ID

The network ID is an identifier shared by the manager and the network motes, and serves to bind motes to the proper network. When a mote “hears” an advertisement containing the correct network ID, it is synchronized to its network manager and able to insert a join request into the network. Dust Networks’ network manager ships with a default network ID. To change the network ID, use the commands described in Table 4.

Note that a mote’s network ID can also be set via the mote serial API, as described in the *IA-510 Mote Serial API Guide*.

Table 4 API Commands for Changing the Network ID

Command	Description
setConfig config/Network/networkId	Changes network ID on the manager. For more information, see “Setting a Configuration Object” on page 19 and “element config/Network/networkId” on page 67.

Table 4 API Commands for Changing the Network ID (Continued)

Command	Description
exchangeNetworkId	Changes network ID on the manager and network motes. For more information, see “exchangeNetworkId” on page 42.

Network Security

The IA-510 network manager *config/Security/securityMode* API command offers a choice of two network security modes—*Accept Common Join Key* and *Accept ACL*. These modes are described in the following sections.

Common Join Key Mode

If the security mode is set to “Accept Common Join Key,” any mote sharing the network manager’s common join key is allowed into the network, as well as any mote on the access control list (see the following section, “ACL Mode”).

The join key is a symmetric encryption key that is shared between the manager and the motes in its network. Dust Networks’ network manager ships with a default common join key, which should be considered public knowledge. The default common join key is typically changed in advance of network deployment, but can also be changed after the network has formed. If the common join key is left unchanged, overhearing and decrypting the packets that assign the session keys is difficult, but technically possible. Therefore, it is highly recommended to change the common join key to a secret one. To change the common join key, use the commands described in Table 5.

Table 5 API Commands for Changing the Common Join Key

Command	Description
setConfig config/Security/commonJoinKey	Changes the common join key on the manager. For more information, see “Setting a Configuration Object” on page 19 and “element config/Security/commonJoinKey” on page 87.
exchangeJoinKey	Changes the common join key on the manager and network motes. For more information, see “exchangeJoinKey” on page 39.

ACL Mode

If the security mode is set to “Accept ACL,” only motes on the manager’s access control list (ACL) are allowed to join the network. If the ACL contains no entries, no motes will

be allowed to join the network. The ACL is managed by the OEM controller, which uses the *createConfig config/Security/Acl* command to add motes and the *deleteConfig config/Security/Acl* command to remove motes from the ACL.

Note that the ACL list can be set up with the same join key for all motes, or a unique join key for each mote. Using the ACL with a unique join key for each mote provides the highest security but requires the most effort on the part of the commissioning workforce to configure the manager and the motes to work together.

The ACL can be configured in advance of network deployment if the MAC address of each network mote is known. If the MAC addresses are not known, the motes may be allowed to join the network using the default common join key and the ACL can be configured after getting the mote MAC addresses from the *netMoteJoin* event notifications that are sent to the manager when a mote joins the network (see “element notifications/event/netMoteJoin” on page 129).

Before adding a new mote to an ACL network, you must first add the mote’s MAC address and join key to the ACL list. When removing a mote from an ACL network, you should remove the mote MAC address from the ACL to preserve network security.

To change the mote join key in Accept ACL mode, use the commands described in Table 6.

Table 6 API Commands for Changing the Mote Join Key

Command	Description
<i>setConfig config/Security/Acl/Device/joinKey</i>	Changes a mote join key on the manager’s ACL list. For more information, see “Setting a Configuration Object” on page 19 and “element config/Security/Acl” on page 87.
<i>exchangeMoteJoinKey</i>	Changes a mote join key on the manager and the specified mote. For more information, see “exchangeMoteJoinKey” on page 40.

SmartMesh Manager XML API Reference

This reference section documents the SmartMesh manager XML API commands and schemas, which client applications use to interact with a SmartMesh manager.

- Network Commands—This includes the commands a client application sends to a SmartMesh manager to configure or query the network.
- Configuration Schema Reference—This section describes the network configuration.
- Notification Schema Reference—This section describes the format of the XML stream sent over the notification channel to a client.

Network Commands

Client applications can incorporate SmartMesh manager XML API commands, making it possible to automate log in and other procedures.

For example, a client application uses the “subscribe” method to send a subscription request to the manager:

```
<methodCall>
<methodName> subscribe </methodName><params>
<param>
  <value><string>dn7c049dc1-3</string></value>
</param>
<param>
  <value><string> all </string></value>
</param></params>
</methodCall>
```

In the example above, a *string* names the element and specifies its content and child elements to the depth specified in the request. A *token* is the token returned to the client when it logs in to the control channel, and used in all subsequent requests.

The following tables document the XML-RPC commands that a client can send to a SmartMesh manager. The commands may return an error message if the input parameters are not valid.

Table 7 Network Command Summary

Network Command	Description
activateAdvertising	Turns on advertising.
activateFastPipe	Turns on a pipe between the manager and a mote.
cancelOtap	Cancels an OTAP in progress.
cli	Tunnels a given CLI command through to the manager's CLI.
createConfig	Creates a configuration object.
deactivateFastPipe	Turns off a pipe between the manager and a mote.
decommissionDevice	Decommissions a mote.
deleteConfig	Deletes a configuration object.
exchangeJoinKey	Updates the join key on the manager and all connected motes in the network using the common join key.
exchangeMoteJoinKey	Updates the join key on a specified mote.
exchangeNetworkId	Updates the network ID and distribute it to motes.
exchangeNetworkKey	Generates a new network key and distribute it to motes.
exchangeSessionKey	Generates a new session key and distribute it to mote(s).
getConfig	Requests configuration information.
getLatency	Requests estimate of data latency for a specified mote.
getTime	Requests the current time.
login	Logs in to the control channel.
logout	Logs out of the control channel.
pingMote	Sends a packet to a mote requesting a response.
reset <object>	Resets an object, such as a mote.
sendRequest	Sends a request packet to a mote.
sendResponse	Sends a response packet to a mote.
setConfig	Sets configuration values.
startOtap <numRetries>	Begins the automatic process of upgrading mote software using over-the-air programming (OTAP).
subscribe	Subscribes to the data notification channel.
unsubscribe	Shuts down a notification TCP connection to a client.

activateAdvertising

This command triggers the manager to turn on advertising for all motes. To confirm the advertising status, wait a few seconds after issuing the command and then issue the command “`element config/Motes/Mote/advertisingStatus`” on page 96.

syntax

```
activateAdvertising <token> <destMacAddress> <timeout>
```

Parameter	Type	Description
token	String	Login token for this session.
destMacAddress	String	ff-ff-ff-ff-ff-ff (activates advertising on all motes)
timeout	Integer	The number of minutes (from 0 to 255) that advertising should remain on. After this period, advertising is turned off. Specifying “0” indicates that the default timeout should be used.

returns

Parameter	Type	Description
result	String	‘OK’ indicates the manager accepted the command and will attempt to turn on advertising. Note that ‘OK’ does not signify that advertising was turned on. An error is returned if command cannot be executed.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_VAL_UNCHANGEABLE

API_OBJECT_NOT_FOUND

activateFastPipe

This command triggers the manager to activate a high-bandwidth connection (pipe) between the manager and a mote. Only one pipe may be active in the network at a time (see ““element notifications/data” on page 112Network Bandwidth Control” on page 25). A notification is sent when the pipe is successfully turned on (see “element notifications/event/netPipeOn” on page 128). Another way to determine the pipe status is to issue the `getConfig config/Motes/Mote/pipeStatus` command (see page 95).

Note that the actual pipe bandwidth allocated by the manager depends on power information reported by the destination mote and the motes through which the pipe is built. To determine the allocated pipe bandwidth, issue the `getConfig config/Motes/Mote/allocatedPipePkPeriod` command (see page 95).

syntax

```
activateFastPipe <token> <destMacAddress>
<pipeDirection>
```

Parameter	Type	Description
token	String	Login token for this session.
destMacAddress	String	MAC address pipe destination.
pipeDirection	Enum	Pipe direction (case sensitive): UniUp = Upstream UniDown = Downstream Bi = Both upstream and downstream

returns

Parameter	Type	Description
result	String	'OK' indicates the manager accepted the command to activate the pipe. ('OK' does not signify that the pipe was activated) An error is returned if command cannot be executed.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_OBJECT_NOT_FOUND

API_VAL_CANT_CREATE

API_VAL_ENUM

API_VAL_UNCHANGEABLE

cancelOtap

This command cancels the OTAP (over-the-air-programming) process to upgrade mote software.

! **Important:** This command is not valid if the OTAP is in the “committing” stage in which motes are switching over to the new software. You can query the OTAP status using the getConfig command (see [“getConfig” on page 45](#)).

syntax

```
cancelOtap <token>
```

Parameter	Type	Description
token	String	Login token for this session.

returns

Parameter	Type	Description
result	String	'OK' indicates automatic OTAP process was cancelled. If OTAP was not cancelled, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_OBJECT_NOT_FOUND

API_NETLAYER_FULL

cli

This command tunnels a given cli command through to the manager’s command line interface (CLI). The cli command can be called by only one XML API client at a time. The response to the given cli command is tunneled back to the client via the notifications channel (see [“element notifications/cli” on page 142](#)). To receive the cli notification, the client must be subscribed to cli notifications (see [“Receiving Data Over the Notification Channel” on page 23](#)).

syntax

```
cli <token> <cli command>
```

Parameter	Type	Description
token	String	Login token for this session.
cli command	String	CLI command to be tunneled.

returns

Parameter	Type	Description
result	String	'OK' indicates that the manager accepted the command for processing. An error is returned if the command was not accepted.

See “Error Messages” on page 57 for error message descriptions.

Error Messages

API_CLI_NULL_COMMAND
API_CLI_WRITE_ERROR
XML_INVALID_AUTHENTICATION

createConfig

This command creates a new element in a list. For a list of the elements that may be used with this command, see [Table 9 on page 60](#).

syntax

```
createConfig <token> <objectPath> <document string>
```

Parameter	Type	Description
token	String	Login token for this session.
objectPath	String	XPath string describing level of new element.
document string	String	XML document describing new element.

returns

Parameter	Type	Description
document string	String	Configuration details with changes incorporated.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_FORMAT

XML_INVALID_DATA

XML_PARSE_ERROR

deactivateFastPipe

This command triggers the manager to start deactivating a high-bandwidth channel between the manager and specified mote. Only one pipe may be active in the network at a time. A notification is sent when the pipe has been successfully turned off (see “[element notifications/event/netPipeOff/macAddr](#)” on page 128).

! **Important:** This command can only deactivate a pipe that was activated by the manager API. Do not call this command to deactivate a pipe that was activated as a result of a service request from a network device.

syntax

```
deactivateFastPipe <token> <destMacAddress>
```

Parameter	Type	Description
token	String	Login token for this session.
destMacAddress	String	MAC address of pipe destination.

returns

Parameter	Type	Description
result	String	'OK' indicates the command was accepted.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_OBJECT_NOT_FOUND

decommissionDevice

This command triggers the manager to prepare a mote (identified by its MAC address) for removal from the network. The manager re-routes traffic to make the mote a leaf node (a node with no other motes reporting to it). When the mote is ready to be removed from the network, a sysManualMoteDecommission event is generated (see “[element notifications/event/sysManualMoteDecommission](#)” on page 119).

syntax

```
decommissionDevice <token> <macAddress>
```

Parameter	Type	Description
token	String	Login token for this session.
macAddress	String	The MAC address of the device to be decommissioned.

returns

Parameter	Type	Description
result	String	'OK' indicates that the command was received and the manager will attempt to decommission the mote.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_OBJECT_NOT_FOUND

deleteConfig

This command deletes an element from the list. For a list of the elements that may be used with this command, see [Table 9 on page 60](#).

syntax

```
deleteConfig <token> <document request>
```

Parameter	Type	Description
token	String	Login token for this session.
document request	String	XML document describing the element to be deleted.

returns

Parameter	Type	Description
result	String	'OK' indicates the element was deleted. If the element was not deleted, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_FORMAT

XML_INVALID_DATA

exchangeJoinKey

This command triggers the manager to initiate a change of common join key for all motes in the network. The join key is a symmetric encryption key that is used by the manager and motes to encrypt and decrypt the join messages that they exchange when the mote is attempting to join the network. The `exchangeJoinKey` command cannot be used if the manager is in `acceptACL` mode or if it is in `acceptCommonJoinKey` mode and there are entries in the access control list (ACL). For more information about the security modes, see “[element config/Security/securityMode](#)” on page 87.

The join key is exchanged without loss of data, and no resetting of motes is needed. The execution of the exchange is set in the future to allow time for the command to propagate across the network and allow time for re-transmissions. The manager generates a `sysCmdFinish` event (see “[element notifications/event/sysCmdFinish](#)” on page 122) when the command is synchronously executed by the motes. The delay is typically tens of minutes after the `exchangeJoinKey` command is issued and depends on the network size and configuration. For networks using the P1 configuration, the delay is around 21 minutes. During this period no additional `exchangeJoinKey` commands may be issued. Motes that reset during the exchange may or may not receive the new join key. If a mote does not receive the new join key it cannot re-join the network.

Note that network motes must be in the Operational state (see “[element config/Motes/Mote/state](#)” on page 94) when the `exchangeJoinKey` command is issued in order for them to receive the new join key. For best results, use the following procedure:

- ❶ Before exchanging the join key, first determine if all network motes are operational by issuing the `getConfig config/Motes/Mote/state` command. If there are non-operational motes, wait for them to rejoin the network. If they do not rejoin within an hour, troubleshoot the problem (for example, check the mote batteries).
- ❷ When all motes are operational, issue the `exchangeJoinKey` command.
- ❸ When the `sysCmdFinish` event is received, reissue the `getConfig config/Motes/Mote/state` command to determine if all motes are operational. Do one of the following depending on the outcome:
 - If all motes are operational, they have all received the new join key and no further action is needed.
 - If some motes are non-operational, wait up to an hour to see if the missing motes can rejoin the network. If the missing motes do not rejoin, repeat steps 2 and 3 with the old network join key so that these motes can rejoin. Then repeat steps 1-3 with the new join key.

syntax

```
exchangeJoinKey <token> <newJoinKey>
```

Parameter	Type	Description
token	String	Login token for this session.
newJoinKey	String	New key as 32-character hexadecimal string (representing 16 bytes).

returns

Parameter	Type	Description
callbackId	Integer	If the command is accepted, the manager returns the callback ID for this request. When the exchange is completed, the manager issues a <code>sysCmdFinish</code> event notification that includes the corresponding callback ID (see “element notifications/event/sysCmdFinish” on page 122). If the command was not accepted by the manager, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION
 API_VAL_UNCHANGEABLE
 API_VAL_ENCKEY
 API_OBJECT_NOT_FOUND

exchangeMoteJoinKey

This command triggers the manager to initiate the exchange of the join key for a specific mote (identified by its MAC address) and update the access control list (ACL). The join key is a symmetric encryption key that is used by the manager and motes to encrypt and decrypt the join messages that they exchange when the mote is attempting to join the network. If the ACL does not contain an entry with this MAC address, the manager creates a new entry. When the exchange is completed, a `sysCmdFinish` event is generated (see [“element notifications/event/sysCmdFinish” on page 122](#)). The join key is exchanged without loss of data, and the mote does not need to be reset.

! **Important:** The time required to complete the command and issue a `sysCmdFinish` notification may vary depending on the size and type of the network. During this period no additional `exchangeMoteJoinKey` commands may be issued.

Note that mote whose join key you wish to change must be in the Operational state (see “[element config/Motes/Mote/state](#)” on page 94) when the `exchangeMoteJoinKey` command is issued in order to receive the new join key. For best results, use the following procedure:

- ❶ Before exchanging the join key, first determine if the mote is operational by issuing the `getConfig config/Motes/Mote/state` command. If the mote is non-operational, wait for it to rejoin the network. If it does not rejoin within an hour, troubleshoot the problem (for example, check the mote batteries).
- ❷ When the mote is operational, issue the `exchangeMoteJoinKey` command.
- ❸ When the `sysCmdFinish` event is received, reissue the `getConfig config/Motes/Mote/state` command to determine if the mote is still operational. Do one of the following depending on the outcome:
 - If the mote is operational, it has received the new join key and no further action is needed.
 - If the mote is non-operational, wait up to an hour to see if it can rejoin the network. If it does not rejoin, repeat steps 2 and 3 with the old join key so that it can rejoin. Then repeat steps 1-3 with the new network join key.

syntax

```
exchangeMoteJoinKey <token> <macAddress> <newJoinKey>
```

Parameter	Type	Description
token	String	Login token for this session.
macAddress	String	The MAC address of the mote to which the manager is distributing the join key.
newJoinKey	String	New key as 32-character hexadecimal string (representing 16 bytes).

returns

Parameter	Type	Description
callbackId	Integer	If the command is accepted, manager returns the callback ID for this request. When the exchange is completed, the manager issues a <code>sysCmdFinish</code> event notification that includes the corresponding callback ID (see “ element notifications/event/sysCmdFinish ” on page 122). If the command was not accepted by the manager, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_VAL_UNCHANGEABLE

API_VAL_ENCKEY

API_OBJECT_NOT_FOUND

exchangeNetworkId

This command initiates an update of the network ID for all motes in the network. The network ID is exchanged without loss of data. The execution of the exchange is set in the future to allow time for the command to propagate across the network and allow time for re-transmissions. The manager generates a `sysCmdFinish` event (see “[element notifications/event/sysCmdFinish](#)” on page 122) when the command is synchronously executed by the motes. The delay is typically tens of minutes after the `exchangeNetworkId` command is issued and depends on the network size and configuration. For networks using the P1 configuration, the delay is around 21 minutes. During this period, no additional `exchangeNetworkId` commands may be issued. After the exchange is completed, the manager and motes must be reset for the new network ID to become effective.

Note that network motes must be in the Operational state (see “[element config/Motes/Mote/state](#)” on page 94) when the `exchangeNetworkId` command is issued in order for them to receive the new network ID. For best results, use the following procedure:

- ❶ Before exchanging the network ID, first determine if all network motes are operational by issuing the `getConfig config/Motes/Mote/state` command. If there are non-operational motes, wait for them to rejoin the network. If they do not rejoin within an hour, troubleshoot the problem (for example, check the mote batteries).
- ❷ When all motes are operational, issue the `exchangeNetworkId` command.
- ❸ When the `sysCmdFinish` event is received, reissue the `getConfig config/Motes/Mote/state` command to determine if all motes are operational. Do one of the following, depending on the outcome:
 - If all motes are operational, they have all received the new network ID. Reset the network.
 - If some motes are non-operational, wait up to an hour to see if the missing motes can rejoin the network. If the missing motes do not rejoin, repeat steps 2 and 3 with the old network ID so that these motes can rejoin. Then repeat steps 1-3 with the new network ID.

syntax

```
exchangeNetworkId <token> <newNetworkId>
```

Parameter	Type	Description
token	String	Login token for this session.
newNetworkId	Integer	New network ID.

returns

Parameter	Type	Description
callbackId	Integer	If the command is accepted, the manager returns the callback ID for this request. When the exchange is completed, the manager issues a sysCmdFinish event notification that includes the corresponding callback ID (see “element notifications/event/sysCmdFinish” on page 122). If the command was not accepted by the manager, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages


XML_INVALID_AUTHENTICATION

API_VAL_MINMAX

API_VAL_UNCHANGEABLE

exchangeNetworkKey

This command triggers the manager to generate a new network key and distribute it to the motes. The network key is exchanged without loss of data, and no resetting of motes is needed. The execution of the exchange is set in the future to allow time for the command to propagate across the network and allow time for re-transmissions. The manager generates a sysCmdFinish event (see [“element notifications/event/sysCmdFinish” on page 122](#)) when the command is synchronously executed by the motes. The delay is typically tens of minutes after the exchangeNetworkKey command is issued and depends on the network size and configuration. For networks using the P1 configuration, the delay is around 21 minutes. During this period no additional exchangeNetworkKey commands may be issued. Note that if a mote resets during the exchange, it will receive the new network key when it re-joins the network.

 **Note:** The network key is a network-wide symmetric authentication key that is shared by the manager and the motes in its network. The network key is used to authenticate all messages other than join requests, activation messages, and advertisements on the data link (MAC) layer. The manager gives the network key to motes when they join the network.

syntax

```
exchangeNetworkKey <token>
```

Parameter	Type	Description
token	String	Login token for this session.

returns

Parameter	Type	Description
callbackId	Integer	If the command is accepted, the manager returns the callback ID for this request. When the exchange is completed, the manager issues a sysCmdFinish event notification that includes the corresponding callback ID (see “element notifications/event/sysCmdFinish” on page 122). If the command was not accepted by the manager, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION
API_VAL_UNCHANGEABLE

exchangeSessionKey

This command triggers the manager to generate and distribute a new random session key for the session between the manager (or gateway) and a mote. The session key is used to encrypt all messages exchanged between the manager (or virtual gateway) and a mote. When the session key exchange is completed, a sysCmdFinish event is generated (see [“element notifications/event/sysCmdFinish” on page 122](#)). The session key is exchanged without loss of data, and no resetting of motes is needed.

! **Important:** The time required to complete the command and issue a sysCmdFinish notification may vary depending on the size and type of the network. During this period no additional exchangeSessionKey commands may be issued.

syntax

```
exchangeSessionKey <token> <macAddress1> <macAddress2>
```

Parameter	Type	Description
token	String	Login token for this session.
macAddress1	String	Identifies the MAC address of the manager (or virtual gateway) for which the session key is to be generated.
macAddress2		Identifies the mote for which the session key is to be generated. The broadcast session key is distributed if you set ff-ff-ff-ff-ff-ff as the mote address.

returns

Parameter	Type	Description
callbackId	Integer	If the command is accepted, the manager returns the callback ID for this request. When the exchange is completed, the manager issues a sysCmdFinish and sysConfigChange event notifications that include the corresponding callback ID (see “element notifications/event/sysCmdFinish” on page 122 , and “element notifications/event/sysConfigChange” on page 121). If the command was not accepted by the manager, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION
 API_VAL_UNCHANGEABLE
 API_OBJECT_NOT_FOUND
 API_NETLAYER_FULL

getConfig

This command requests configuration details for a specified element. For a list of the elements that may be used with this command, see [Table 9 on page 60](#). Note that if alarms or notifications are requested and none are present, the error message “OBJECT NOT FOUND” is returned.

syntax

```
getConfig <token> <depth> <document request>
```

Parameter	Type	Description
token	String	Login token for this session.
depth	String	Specifies number of levels of child elements to return.
document request	String	XML document describing requested element.

returns

Parameter	Type	Description
document string	String	Configuration details.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

API_OBJECT_NOT_FOUND

XML_INVALID_FORMAT

XML_INVALID_DATA

XML_PARSE_ERROR

API_VAL_MINMAX

getLatency

This command returns an estimate of the latency to communicate with a specific mote (identified by its MAC address). The return value is an array of two integers representing upstream latency and downstream latency, measured in milliseconds. The estimate applies only to the packet’s over-the-air time, and does not include time it may spend in manager’s queues prior to being sent. After the manager sends the packet into the network, it issues a netPacketSent event notification that includes the callback ID (see “element notifications/event/netPacketSent” on page 134).

Note that the latency calculations take into account long-term bandwidth, such as bandwidth set by the manager API or requested by service requests, but not short-term bandwidth or pipe bandwidth.

syntax

```
getLatency <token> <macAddress>
```

Parameter	Type	Description
token	String	Login token for this session.
macAddress	String	MAC address of the mote.

returns

Array of {downstream latency, upstream latency}

Parameter	Type	Description
downstream latency	Integer	Data latency (in milliseconds) for transmissions from the manager to mote.
upstream latency	Integer	Data latency (in milliseconds) for transmissions from mote to the manager.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

API_OBJECT_NOT_FOUND

getTime

This command requests the current time as Coordinated Universal Time (UTC) and ASN (absolute slot number).

syntax

```
getTime <token>
```

Parameter	Type	Description
token	String	Login token for this session.

returns

Array of {UTCtime, ASNtime}

Parameter	Type	Description
UTCtime	String	Time as seconds.microseconds.
ASNtime	String	Time as absolute slot number (ASN).

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

API_OBJECT_NOT_FOUND

login

This command logs a client in to the manager’s control channel. The default username is “admin” and default password is “admin”.

syntax

```
login <username> <password>
```

Parameter	Type	Description
username	String	Your username.
password	String	Your password.

returns

Parameter	Type	Description
token	String	Control token used in all subsequent commands.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

logout

This command logs a client out of the control channel (does not log out of notification channel).

syntax

```
logout <token>
```

Parameter	Type	Description
token	String	Login token for this session.

returns

Parameter	Type	Description
result	String	'OK' indicates the logout was completed. If logout was not completed, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

pingMote

This command sends a packet to a mote requesting a response. When the manager receives a reply from the mote, it will issue a netPingReply event notification that includes this callback ID (“element notifications/event/netPingReply” on page 138).

syntax

```
pingMote <token> <macAddress>
```

Parameter	Type	Description
token	String	Login token for this session.
macAddress	String	The MAC address of the mote you want to ping.

returns

Parameter	Type	Description
callbackId	Integer	If the packet is accepted, the manager returns the callback ID for this request. If the packet was not accepted by the manager, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_OBJECT_NOT_FOUND

API_NETLAYER_FULL

reset <object>

This command resets (or clears) an object such as a network or a system. When the mote is the object of the reset command, either a mote ID or MAC address may be used to identify the mote. The reset mote command is always sent as a best-effort packet. A notification message (*notification/event/sysManualMote/Reset*) is issued when the manager sends the reset command to a mote. To verify that the mote has received the command and reset itself, use the *getConfig config/Motes/Mote/state* command to check the mote state.

syntax

```
reset <token> <object>
reset <token> mote {<moteId>|<macAddress>}
```

Parameter	Type	Description
token	String	Login token for this session.
object (system, network, stat, eventLog, mote)	String	Identifies the object to be reset: system Resets system software. network Resets the network. stat Resets statistics. eventLog Resets the eventLog. mote Resets the mote.

returns

Parameter	Type	Description
result	String	`OK` indicates the object was reset for all objects except a mote. If the object was not reset, an error message is returned. For a mote, `OK` indicates the reset command was sent to the mote.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

XML_PARSE_ERROR

API_OBJECT_NOT_FOUND

API_NETLAYER_FULL

sendRequest

This command may be used to send a packet to a specific mote (identified by its MAC address) or all motes. The packet may either be the request packet to initiate request/response exchange with the mote (*reliable* is “true”) or an unacknowledged packet (*reliable* is “false”). *Note that broadcast packets must be sent as unacknowledged packets.*

The binary data is sent as a string in which each byte is represented as a two-digit hexadecimal number. The maximum data payload is 78 bytes. *Note that messages must prepend the 4-byte header, 00 00 FC 12, leaving 74 usable bytes.*

If manager’s internal queue is full, it responds with the error message API_NETLAYER_FULL. The application should retry after a short timeout.

syntax

```
sendRequest <token> <macAddress> <application domain>
<priority> <reliable> <data>
```

Parameter	Type	Description
token	String	Login token for this session.
macAddress	String	MAC address of mote to which the packet is to be sent. If you set the MAC address to ff-ff-ff-ff-ff-ff, the packet is broadcast to all motes.
application domain	String	Indicates the type of packet to be sent to the mote. publish Data packets sent on a regular, periodic basis. event Packets (such as alarms) that are triggered by an event. maintenance A series of request/response packets used to manage the device.
priority	String	Indicates the packet priority. high The packet is high priority. medium The packet is medium priority. low The packet is low priority.

Parameter	Type	Description
reliable	Boolean	“True” if this is part of an acknowledged transaction. “False” if this is part of a best-effort (unacknowledged) transaction. Note that a broadcast packet must be sent as best-effort transaction.
data	String	Packet payload, sent as a hexadecimal string. Maximum 74 bytes.

returns

Parameter	Type	Description
callbackId	Integer	<p>If the packet is accepted by the manager and queued for transmission, the manager returns the callbackId for this request. After the manager sends the packet into the network, it issues a netPacketSent event notification that includes the callback ID (see “element notifications/event/netPacketSent” on page 134).</p> <p>If the sendRequest is a reliable transaction, the manager also includes the callbackId when it sends the client the reply data packet it receives from the mote. If the sendRequest is a best-effort transaction, no callbackId is provided with the reply.</p> <p>If the packet was not accepted, an error message is returned.</p>

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_VAL_ENUM

API_OBJECT_NOT_FOUND

API_NETLAYER_FULL

API_VAL_CROSSREF

API_VAL_PATTERN_HEX

API_VAL_DATAPACKET

sendResponse

This command allows the client to send a response to a reliable request generated by a mote (identified by its MAC address). The binary data is sent as a string in which each byte is represented as a two-digit hexadecimal number. For maximum data payload, see “sendRequest” on page 50. To be properly associated with the request packet, the response must contain the *callbackId* of the request packet. For more information, see [“element notifications/data/dataInfo/callbackId” on page 113](#).

syntax

```
sendResponse <token> <macAddress> <application domain>
<priority> <reliable> <callbackId> <data>
```

Parameter	Type	Description
token	String	Login token for this session.
macAddress	String	MAC address of mote to which packet is to be sent.
application domain	String	Indicates the type of packet to be sent to the mote. <ul style="list-style-type: none"> publish Data packets sent on a regular, periodic basis. event Packets (such as alarms) that are triggered by an event. maintenance A series of request/response packets used to manage the device.
priority	String	Indicates the packet priority. <ul style="list-style-type: none"> high The packet is high priority. medium The packet is medium priority. low The packet is low priority.
reliable	Boolean	Set to "True".
callbackId	Integer	The callbackId of the original request packet.
data	String	Packet payload, sent as a hexadecimal string. Maximum 74 bytes.

returns

Parameter	Type	Description
result	String	'OK' the packet is accepted by the manager and queued for transmission.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION
 API_OBJECT_NOT_FOUND
 API_VAL_ENUM
 API_NETLAYER_FULL
 API_VAL_CROSSREF
 API_VAL_PATTERN_HEX
 API_VAL_DATAPACKET

setConfig

This command specifies a value for a configuration object. For a list of the elements that may be used with this command, see [Table 9 on page 60](#).

syntax

```
setConfig <token> <document string>
```

Parameter	Type	Description
token	String	Login token for this session.
document string	String	XML document describing the changes to be made.

returns

Parameter	Type	Description
document string	String	Configuration details with changes incorporated.

error messages


See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_FORMAT
 XML_INVALID_DATA
 XML_PARSE_ERROR

startOtap <numRetries>

This command initiates the OTAP (over-the-air-programming) process to upgrade mote software.

 **Important!** Before using the startOtap command, you need to copy the software upgrade onto the manager's /root/otap directory. For example, you may use the manager's Admin Toolset utility for this task (see the "Software Upgrade" section in the *Admin Toolset Guide*).

syntax

```
startOtap <token> <numberOfRetries>
```

Parameter	Type	Description
token	String	Login token for this session.
numRetries	Integer	Number of times the manager attempts to resend otap file into the network. If numRetries is omitted, a default of 100 retries is used.

returns

Parameter	Type	Description
result	String	'OK' indicates automatic OTAP process was started. If OTAP was not started, an error message is returned.

error messages

See "Error Messages" on page 57 for error message descriptions.

Error Messages

XML_INVALID_AUTHENTICATION

API_OBJECT_NOT_FOUND

API_VAL_STATE

subscribe

This command subscribes a client to the notification channel (issued over the control channel).

syntax

```
subscribe <token> <filter>
```

Parameter	Type	Description																				
token	String	<p>Login token for this session.</p> <p>If the token is a control channel login token, a new notification channel is created.</p> <p>If the token is a notification channel token, the subscription filter for the specified notification channel is updated.</p>																				
filter	String	<p>Specifies the notification types to filter through. Any combination of the following filters can be used:</p> <table border="1"> <thead> <tr> <th>Filter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>event-alarm</td> <td>Alarm open and close events</td> </tr> <tr> <td>event-network</td> <td>Network events (such as link add, mote join)</td> </tr> <tr> <td>event-system</td> <td>System events (such as config change, reset dcc)</td> </tr> <tr> <td>data</td> <td>All data packets</td> </tr> <tr> <td>sysError</td> <td>System error messages</td> </tr> <tr> <td>log</td> <td>Log messages</td> </tr> <tr> <td>cli</td> <td>CLI notifications</td> </tr> </tbody> </table> <p>The following shortcuts can also be used with a filter:</p> <table border="1"> <tbody> <tr> <td>events</td> <td>All events</td> </tr> <tr> <td>all</td> <td>All notifications</td> </tr> </tbody> </table> <p>Filters are space delimited.</p>	Filter	Description	event-alarm	Alarm open and close events	event-network	Network events (such as link add, mote join)	event-system	System events (such as config change, reset dcc)	data	All data packets	sysError	System error messages	log	Log messages	cli	CLI notifications	events	All events	all	All notifications
Filter	Description																					
event-alarm	Alarm open and close events																					
event-network	Network events (such as link add, mote join)																					
event-system	System events (such as config change, reset dcc)																					
data	All data packets																					
sysError	System error messages																					
log	Log messages																					
cli	CLI notifications																					
events	All events																					
all	All notifications																					

For example, the following command requests a subscription to all data packets and alarm open and close events:

```
<methodCall>
  <methodName>subscribe</methodName>
  <params>
    <param><value><string>dn7c049dc1-3</string></value></param>
    <param><value><string>data event-alarm</string></value></param>
  </params>
</methodCall>
```

returns

Array of {token, port}

Parameter	Type	Description
token	String	Token for notification channel.
port	Integer	TCP port for notification channel.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

API_VAL_ENUM
XML_INVALID_NOTIF_CLIENT
XML_TOO_MANY_NOTIF_CLIENTS

unsubscribe

This command shuts down a notification channel connection to a client. This command is an alternative to the client closing the notification channel.

syntax

```
unsubscribe <token>
```

Parameter	Type	Description
token	String	Notification token for this session.

returns

Parameter	Type	Description
result	String	'OK' indicates the notification channel TCP connection was closed. If the connection was not closed, an error message is returned.

error messages

See “Error Messages” on page 57 for error message descriptions.

Error Messages

XML_INVALID_NOTIF_CLIENT

Error Messages

Table 8 describes the error message that may be returned by a command.

Table 8 Error Messages

Error Code	Error Message	Description
0	API_OK	The application layer has processed the command successfully.
1	API_END_OF_LIST	The iteration has reached the end of the list of objects.
2	API_OBJECT_NOT_FOUND	The object was not found.
3	API_VAL_CROSSREF	Validation error. The cross reference failed
4	API_VAL_ENUM	Validation error. The value was not in the enumeration.
5	API_VAL_PATTERN	Validation error. The value contained one of the following invalid characters: @, &, <, or >.
6	API_VAL_PATTERN_HEX	Validation error. The value contained an invalid hexadecimal character (valid characters are a-f, A-F, and 0-9).
7	API_VAL_ENCKEY	Validation error. The encryption key must contain exactly 32 hex characters.
8	API_VAL_DATAPACKET	Validation error. The message does not fit in the packet.
9	API_VAL_REQFIELD	Validation error. A required field is missing.
10	API_VAL_LISTLEN	Validation error. Cannot add the new object because the limit has been reached.
11	API_VAL_UNCHANGEABLE	Validation error. The element cannot be changed.
12	API_NOTUNIQCMD	A similar command is already being processed.
13	API_VAL_STATE	Validation error. The current state does not allow this action to be performed.
14	API_VAL_MINMAX	Validation error. The maximum value must be greater than the minimum value.
15	API_VAL_TIMEBOUNDARY	Validation error. The time is not on the required 15-minute boundary.
16	API_VAL_STARTEND	Validation error. The end time must be after start time.
17	API_VAL_LAST_ELEMENT	Validation error. Cannot delete the last element.
18	API_NETLAYER_FULL	The manager's transmit queue is full.

Table 8 Error Messages (Continued)

Error Code	Error Message	Description
19	API_VAL_CANT_CREATE	Validation error. Cannot create the object because it already exists.
20	API_CLI_NULL_COMMAND	Empty CLI command.
21	API_CLI_WRITE_ERROR	Could not write to the CLI session.
22	API_INVALID_MOTE	The specified mote is invalid.
23	API_INVALID_COMMAND	The command is invalid.
24	API_INVALID_ARGUMENT	The argument to the command is invalid.
323	API_GENERAL_ERROR	A general error that does not fall into any other error category.
1000	API_UNIDENTIFIED	Unidentified error.
1001	XML_INVALID_FORMAT	The XML format is invalid.
1002	XML_INVALID_DATA	The XML data is invalid.
1003	XML_PARSE_ERROR	A parsing error occurred.
1004	XML_INVALID_RPC CONNECTION	Internal error.
1005	XML_NVALID_AUTHENTICATION	The command could not be authenticated.
1006	XML_INVALID_NOTIF_CLIENT	The notification client does not exist.
1007	XML_TOO_MANY_NOTIF_CLIENTS	The maximum number of notification clients has been reached.

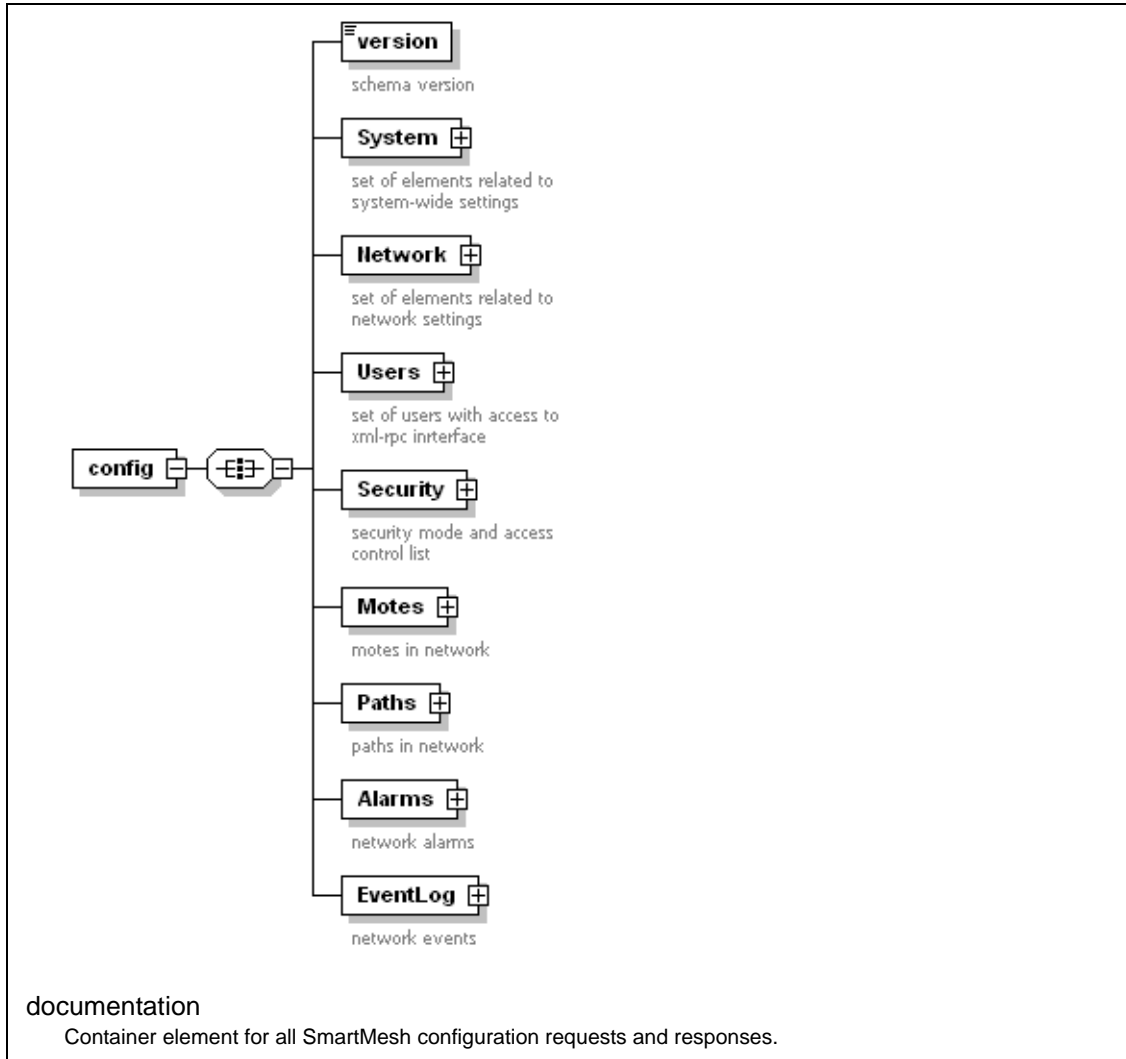
Configuration Schema Reference

This section documents the XML schema that describes the configuration document for a SmartMesh-enabled network. The configuration document is an instance of the schema, DPI.xsd. The manager updates the document in response to configuration commands from clients.

The XML schema contains the following elements:

- “element config”
- “element config/System”
- “element config/Network”
- “element config/Users”
- “element config/Security”
- “element config/Motes”
- “element config/Paths”
- “element config/Alarms”
- “element config/EventLog”

element **config**



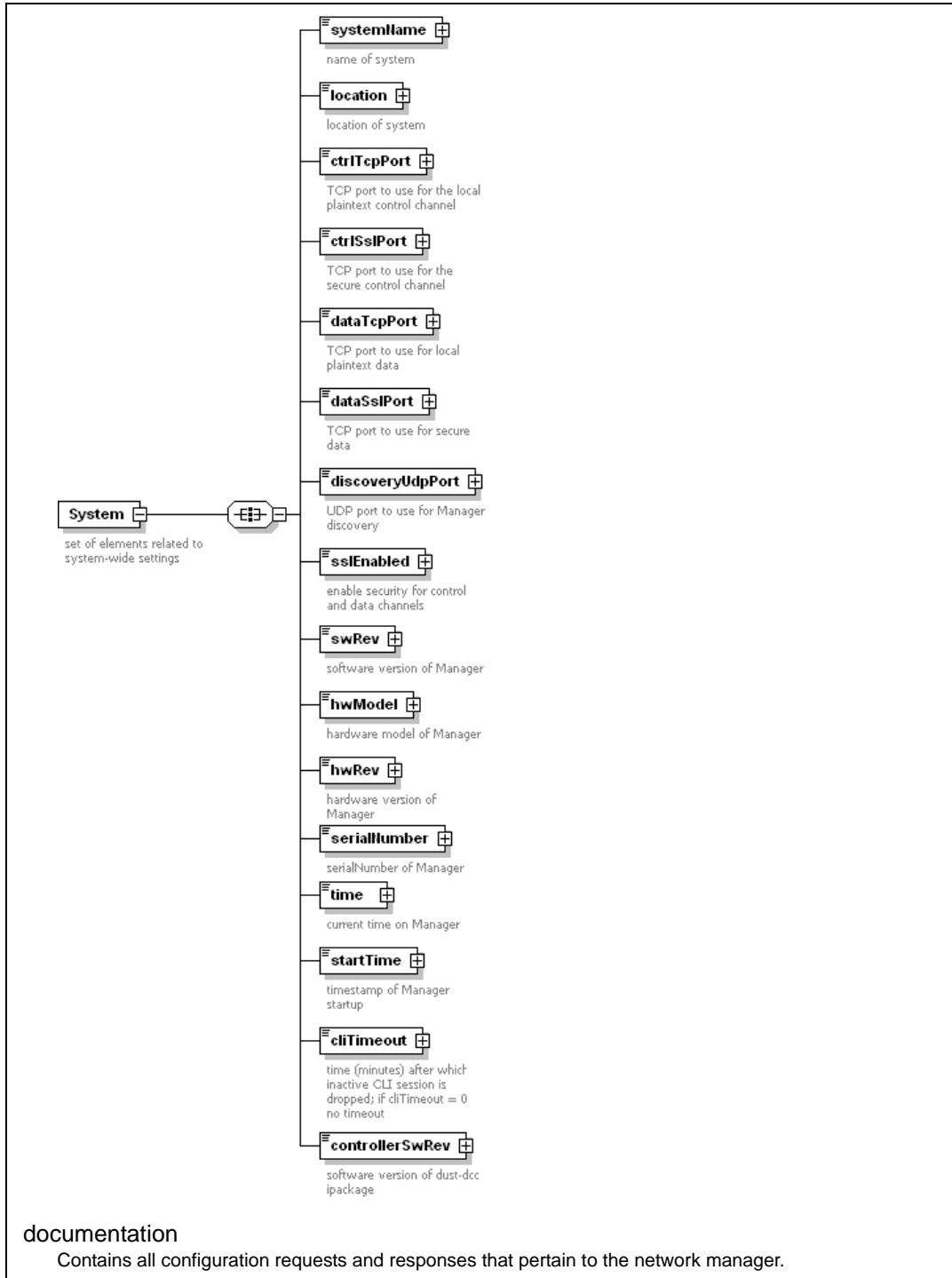
Elements Used in Configuration Commands

Table 9 indicates the configuration elements that may be used in *createConfig*, *deleteConfig*, *setConfig*, and *getConfig* commands.

Table 9 Elements Used in the Config Commands

Element	createConfig	deleteConfig	setConfig	getConfig
System			X	X
Network			X	X
Sla			X	X
OtapStatus				X
Statistics				X
Channel Blacklist			X	X
Users	X	X	X	X
Security			X	X
Acl	X	X	X	X
Motes		X	X	X
Statistics				X
Paths				X
Statistics				X
Alarms				X
Event Log				X

element **config/System**



element `config/System/systemName`

type: xsd:string attributes: read/write
properties minLength: 0 maxLength: 16
documentation Name of the network manager.

element `config/System/location`

type: xsd:string attributes: read/write
properties minLength: 0 maxLength: 16
documentation Location of system.

element `config/System/ctrlTcpPort`

type: xsd:unsignedShort attributes: read/write
properties minInclusive: 1024 maxInclusive: 49000
documentation TCP port to use for the local plaintext control channel. The default port is 4445 (see “Logging In to the Control Channel” on page 12).

element `config/System/ctrlSslPort`

type: xsd:unsignedShort attributes: read/write
properties minInclusive: 1024 maxInclusive: 49000
documentation TCP port to use for the secure control channel. See “Logging In to the Control Channel” on page 12 .

element config/System/dataTcpPort

type: xsd:unsignedShort
attributes: read/write

properties

minInclusive: 1024
maxInclusive: 49000

documentation

TCP port to use for local plaintext notification channel.

element config/System/dataSslPort

type: xsd:unsignedShort
attributes: read/write

properties

minInclusive: 1024
maxInclusive: 49000

documentation

TCP port to use for secure notification channel.

element config/System/discoveryUdpPort

type: xsd:unsignedShort
attributes: read/write

properties

minInclusive: 1024
maxInclusive: 49000

documentation

UPD port to use for network manager discovery.

element config/System/sslEnabled

type: xsd:boolean
attributes: read/write

documentation

Enable security for control and data channels.

If sslEnabled is true, the ctrlTcpPort and dataTcpPort are only accessible from the local host, and SSL is used to provide secure access to the network manager through the ctrlSslPort and dataSslPort.

If sslEnabled is false, ctrlTcpPort and dataTcpPort provide plaintext access for any client.

The sslEnabled toggle only takes effect when the manager restarts.

element config/System/swRev

type: xsd:string attributes: read-only
properties minLength: 0 maxLength: 16
documentation The Dust Networks Controller ipackage version. The ipackage contains the Dust Networks Controller executable and other files such as Admin Toolset, setup scripts, and other Controller modules.

element config/System/hwModel

type: xsd:string attributes: read-only
properties minLength: 0 maxLength: 16
documentation Hardware model of the network manager.

element config/System/hwRev

type: xsd:string attributes: read-only
properties minLength:0 maxLength:16
documentation Hardware version of the network manager.

element config/System/serialNumber

type: xsd:string attributes: read-only
properties minLength:0 maxLength:16
documentation Serial number of the network manager.

element config/System/time

type: dustTime
attributes: read-only

documentation

Current time on the network manager. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element config/System/startTime

type: dustTime
attributes: read-only

documentation

Timestamp of the network manager startup. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element config/System/cliTimeout

type: xsd:unsignedShort
attributes: read/write

properties

default: 120
minInclusive:0
maxInclusive:65535

documentation

Time (minutes) after which inactive command-line interface session is dropped; if cliTimeout = 0, no timeout.

element config/System/controllerSwRev

type: xsd:string
attributes: read-only

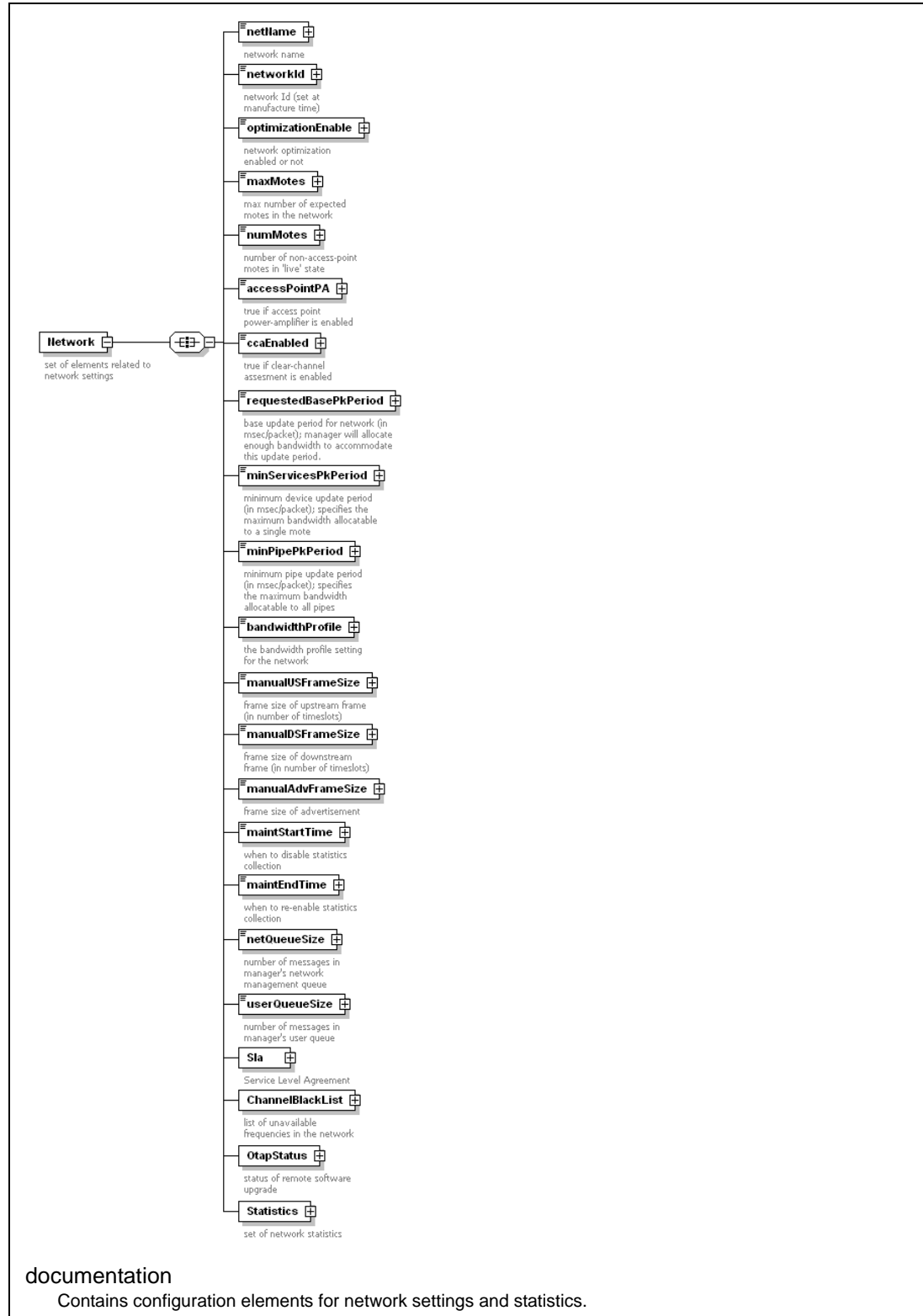
properties

minLength:0
maxLength:16

documentation

The version of the Dust Networks Controller's executable

element **config/Network**



element `config/Network/netName`

type: xsd:string attributes: read/write
properties minLength: 0 maxLength: 16
documentation The network name is a user readable text field to describe the specific network installation.

element `config/Network/networkId`

type: xsd:unsignedShort attributes: read/write
properties minInclusive: 0 maxInclusive: 65535
documentation Network ID (set at time of manufacture).

element `config/Network/optimizationEnable`

type: xsd:boolean attributes: read/write
properties default: true
documentation Enable network optimization. Dust Networks strongly recommends that optimization remain enabled at all times.

element `config/Network/maxMotes`

type: xsd:unsignedShort attributes: read/write
properties minInclusive: 1 maxInclusive: 500
documentation Maximum number of motes expected in the network. Once this limit is reached, no additional motes are allowed to join the network. Be sure to remove motes that are no longer in the network. To delete a mote, use the command "deleteConfig" on page 38 and specify <code><config><Motes><Mote><macAddress></code> as the document request.

element config/Network/numMotes

type: xsd:unsignedShort
attributes: read-only

properties

minInclusive: 0
maxInclusive: 65535

documentation

Number of non-access point motes in Live state.

element config/Network/accessPointPA

type: xsd:boolean
attributes: read/write

documentation

“True” if access point power amplifier is enabled.
This parameter has no effect on products that do not have a power amplifier.

element config/Network/ccaEnabled

type: xsd:boolean
attributes: read/write

documentation

Enables or disables clear channel assessment for all motes in the network, including the access point mote.

element config/Network/requestedBasePkPeriod

type: xsd:unsignedLong
attributes: read/write

properties

minInclusive: 0
maxInclusive: 4294967295

documentation

Defines the base bandwidth (in msec/packet) that the manager should allocate for each device. Note that the config/Network/requestedBasePkPeriod parameter applies to manager-controlled bandwidth, which is independent from bandwidth requested through mote service requests. It does not include bandwidth requested through mote service requests and cannot be used to allocate bandwidth for services.

element `config/Network/minServicesPkPeriod`

type: xsd:unsignedLong attributes: read/write
properties minInclusive: 0 maxInclusive: 4294967295
documentation Defines the maximum bandwidth (in msec/packet) that a single mote may be allocated for total non-pipe user requested bandwidth. This limits service requests from motes.

element `config/Network/minPipePkPeriod`

type: xsd:unsignedLong attributes: read/write
properties minInclusive: 0 maxInclusive: 4294967295
documentation Limits bandwidth (msec/packet) on all pipes (both manager-API requested pipes and pipes as a result of service requests). This is most useful for regulating service requests from field devices.

element `config/Network/bandwidthProfile`

type: xsd:string attributes: read/write
properties enumeration: Manual enumeration: P1 enumeration: P2
documentation The bandwidth profile determines the frame size (number of timeslots) for upstream, downstream, and advertising traffic. There are three profiles available—a normal profile (P1), a low-power profile (P2), and a manual profile, which allows you to set the number of timeslots for upstream, downstream, and advertising traffic. The new bandwidth profile setting takes effect the next time the network reforms. Use the <code>reset<object></code> command (see page 49) to reset the network and cause it to reform. Caution: The manual bandwidth profile is an advanced feature to be used only with direct support from Dust Networks applications engineering.

element `config/Network/manualUSFrameSize`

type: xsd:unsignedLong attributes: read/write
properties minInclusive: 0 maxInclusive: 4294967295
documentation Manually sets the frame size (in number of timeslots) for upstream traffic. This parameter is only applicable if “element config/Network/bandwidthProfile” is set to “manual.”

element `config/Network/manualDSFrameSize`

type: xsd:unsignedLong attributes: read/write
properties minInclusive: 0 maxInclusive: 4294967295
documentation Manually sets the frame size (in number of timeslots) for downstream traffic. This parameter is only applicable if “element config/Network/bandwidthProfile” is set to “manual.”

element `config/Network/manualAdvFrameSize`

type: xsd:unsignedLong attributes: read/write
properties minInclusive: 0 maxInclusive: 4294967295
documentation Manually sets the frame size (in number of timeslots) for advertisement traffic. This parameter is only applicable if “element config/Network/bandwidthProfile” is set to “manual.”

element `config/Network/maintStartTime`

type: dustTime attributes: read/write
documentation Defines when to disable statistics collection. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element config/Network/maintEndTime

type: dustTime
attributes: read/write

documentation

Defines when to re-enable statistics collection. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element config/Network/netQueueSize

type: xsd:unsignedShort
attributes: read-only

properties

minInclusive: 0
maxInclusive: 65535

documentation

Number of messages in the manager's network management queue.

element config/Network/userQueueSize

type: xsd:unsignedShort
attributes: read-only

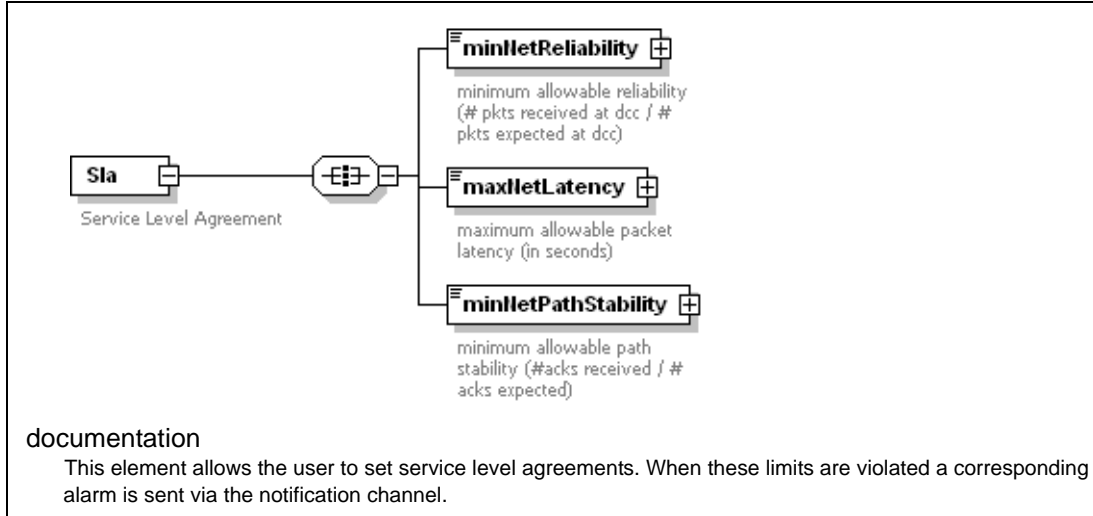
properties

minInclusive: 0
maxInclusive: 65535

documentation

Number of messages in the manager's user queue.

element `config/Network/Sla`



element `config/Network/Sla/minNetReliability`

type: xsd:float
attributes: read/write

documentation

Minimum allowable reliability (the ratio of number of packets received at the manager to the number of packets expected at the manager).

element `config/Network/Sla/maxNetLatency`

type: xsd:unsignedLong
attributes: read/write

documentation

Maximum allowable packet latency (milliseconds).

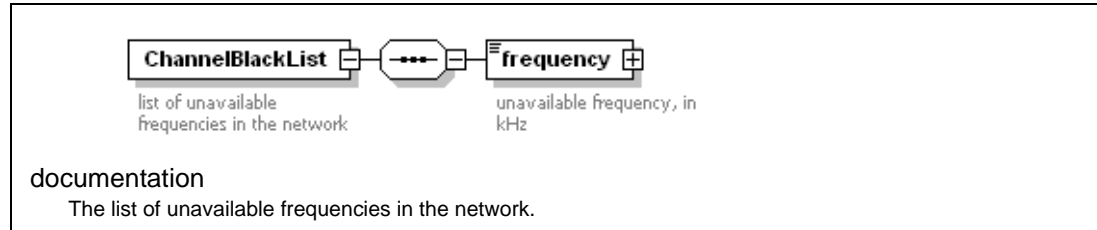
element `config/Network/Sla/minNetPathStability`

type: xsd:float
attributes: read/write

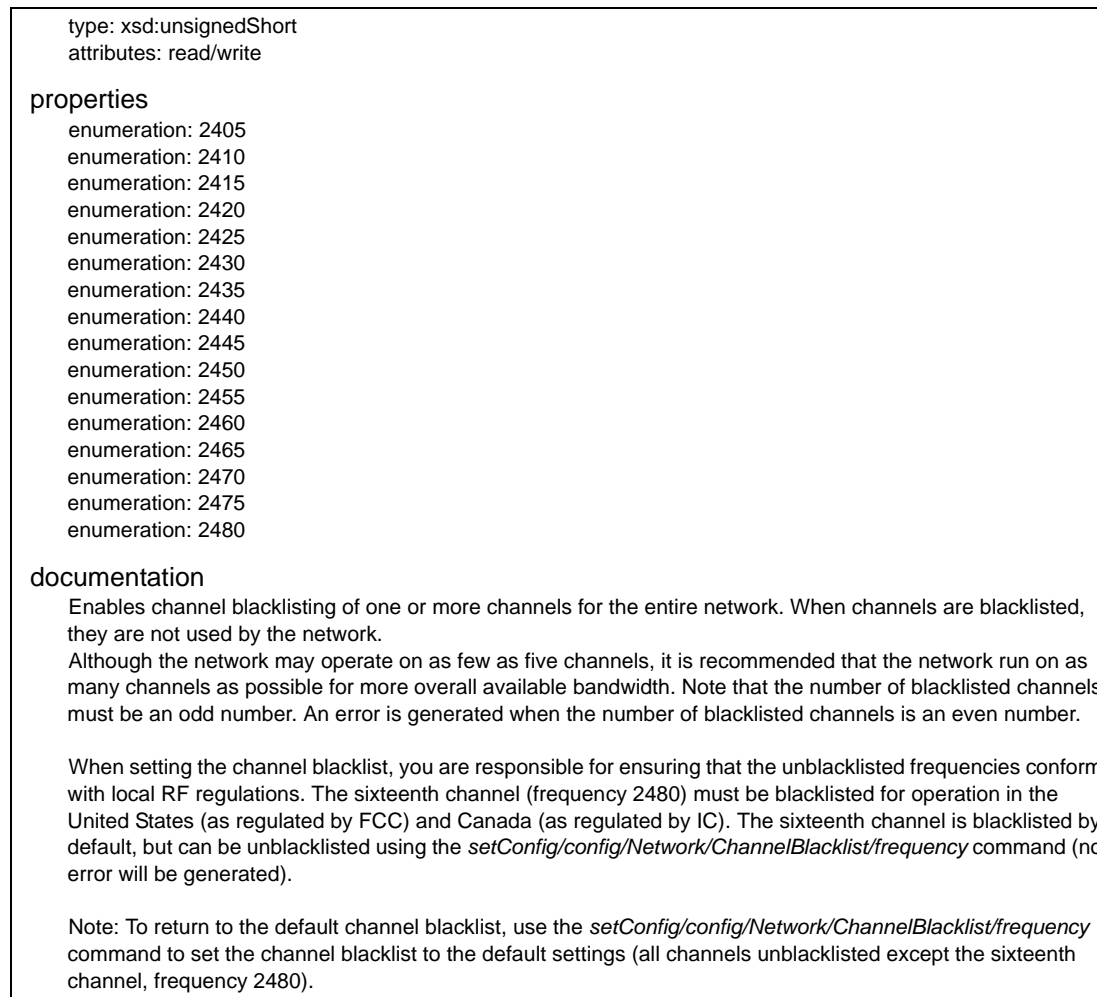
documentation

Minimum allowable path stability (number of acknowledgments received/number of acknowledgments expected).

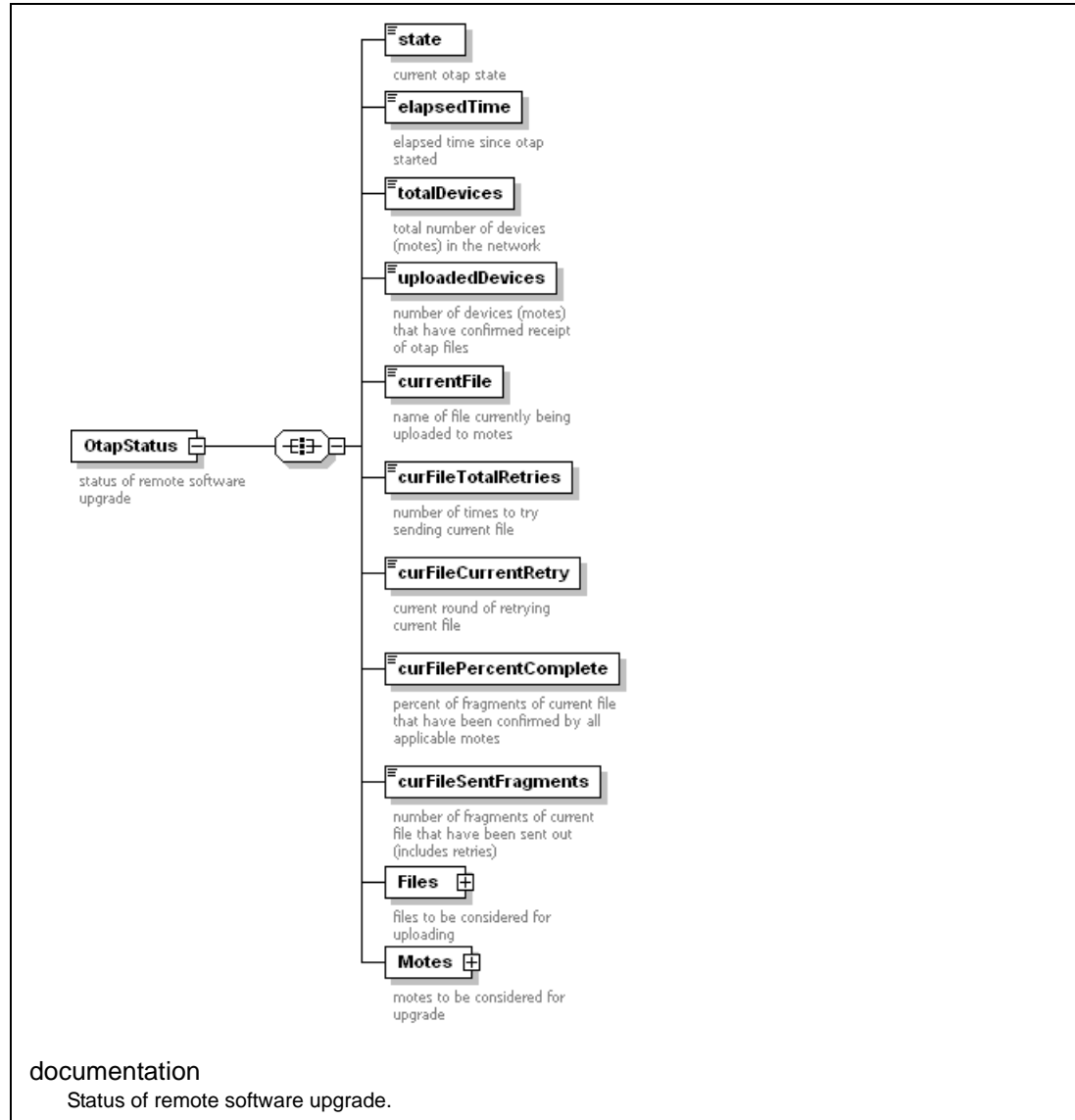
element `config/Network/ChannelBlackList`



element `config/Network/ChannelBlackList/frequency`



element **config/Network/OtapStatus**



element `config/Network/OtapStatus/state`

type: xsd:string
 attributes: read-only

properties

enumeration: Running
 enumeration: Uploading
 enumeration: Committing
 enumeration: Completed

documentation

Current OTAP state.
 Idle indicates that the OTAP process has been cancelled.
 Initializing indicates that OTAP handshakes are being performed in preparation for uploading files to the motes.
 Upload indicates OTAP files are being uploaded to the motes.
 Commit indicates that the upload is now complete and motes are being reset in order to activate the new software. Cancelling an OTAP at this stage will result in motes running different software versions.
 Completed means the OTAP is finished.

element `config/Network/OtapStatus/elapsedTime`

type: dustTime
 attributes: read-only

documentation

Elapsed time since OTAP started. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element `config/Network/OtapStatus/totalDevices`

type: xsd:unsignedShort
 attributes: read-only

properties

minInclusive: 0
 maxInclusive: 65535

documentation

Total number of motes in the network.

element `config/Network/OtapStatus/uploadedDevices`

type: xsd:unsignedShort
 attributes: read-only

properties

minInclusive: 0
 maxInclusive: 65535

documentation

Number of motes that have confirmed receipt of OTAP files.

element config/Network/OtapStatus/currentFile

type: xsd:string attributes: read-only
properties minLength:0 maxLength:32
documentation Name of file currently being uploaded to notes.

element config/Network/OtapStatus/curFileTotalRetries

type: xsd:unsignedShort attributes: read-only
properties minInclusive: 0 maxInclusive: 65535
documentation Number of times to try sending current file.

element config/Network/OtapStatus/curFileCurrentRetry

type: xsd:unsignedShort attributes: read-only
properties minInclusive: 0 maxInclusive: 65535
documentation Current round of retrying current file.

element config/Network/OtapStatus/curFilePercentComplete

type: xsd:float attributes: read-only
documentation Percent of fragments of current file that have been confirmed by all applicable notes.

element **config/Network/OtapStatus/curFileSentFragments**

type: unsignedShort
 attributes: read-only

properties
 minInclusive: 0
 maxInclusive: 65535

documentation
 Number of fragments of current file that have been sent out (includes retries).

element **config/Network/OtapStatus/Files**

The diagram shows a box labeled 'Files' connected to a box labeled 'filename'. Below 'Files' is the text 'files to be considered for uploading'. Below 'filename' is the text 'name of file'.

documentation
 Files to be considered for uploading.

element **config/Network/OtapStatus/Files/fileName**

type: xsd:string
 attributes: read-only

properties
 minLength: 0
 maxLength: 16

documentation
 Name of file to be considered for uploading.

element **config/Network/OtapStatus/Motes**

The diagram shows a box labeled 'Motes' connected to a box labeled 'Mote'. Below 'Motes' is the text 'motes to be considered for upgrade'. Below 'Mote' is the text 'mote to be considered for upgrade'. To the right of 'Mote' is a list of four sub-elements: 'moteld' (mote ID (read-only)), 'macAddr' (mac address (read-only)), 'percentComplete' (percent of fragments of applicable file that have been confirmed as received), and 'status' (current mote otap status).

documentation
 Motes to be considered for upgrade.

element config/Network/OtapStatus/Motes/Mote/moteld

type: xsd:unsignedShort
attributes: read-only

properties

minInclusive: 1
maxInclusive: 528

documentation

Motes to be considered for upgrade.

element config/Network/OtapStatus/Motes/Mote/macAddr

type: xsd:string
attributes: read-only

documentation

Mote MAC address.

element config/Network/OtapStatus/Motes/Mote/percentComplete

type: xsd:float
attributes: read-only

documentation

Percent of fragments of applicable files that have been confirmed as received by the mote.

element `config/Network/OtapStatus/Motes/Mote/status`

type: xsd:string
attributes: read-only

properties

enumeration: NotInOtap
enumeration: InProgress
enumeration: Cancelled
enumeration: LockedOut
enumeration: LowBattery
enumeration: NoSpace
enumeration: FileError
enumeration: MicError
enumeration: Finished

documentation

Current mote OTAP status:

NotInOtap indicates the mote is not receiving OTAP at this moment because the OTAP file does not apply to this mote.

InProgress indicates the OTAP is in progress on this mote.

Cancelled indicates the OTAP was cancelled for this mote.

LockedOut indicates the OTAP will not be performed on this mote because OTAP lockout was set for this mote.

LowBattery indicates the mote battery power is too low for OTAP to be performed.

NoSpace indicates an unexpected error. There is not enough space in the mote flash for the OTAP file.

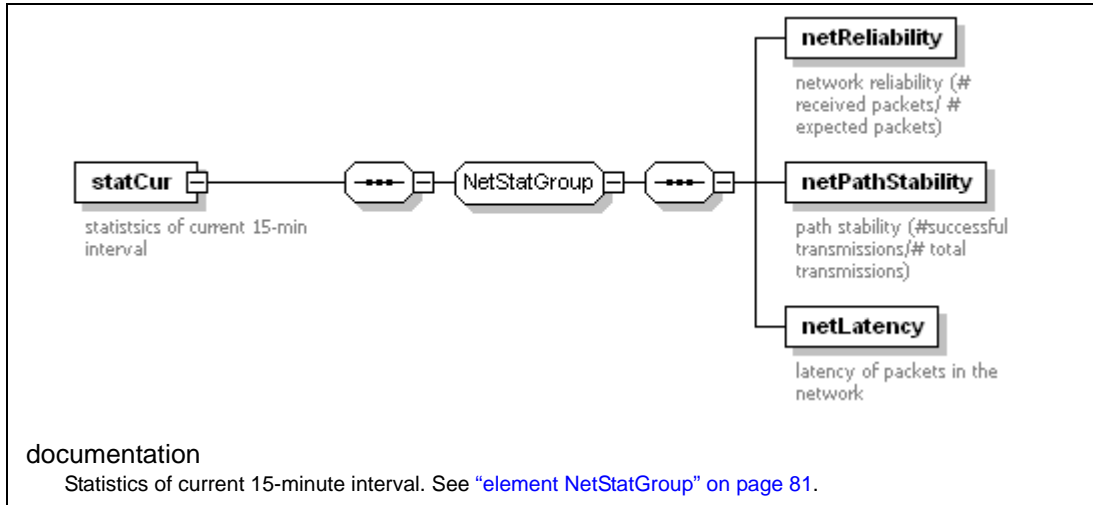
FileError indicates an unexpected error. The OTAP file is invalid or cannot be recognized by the mote.

MicError indicates an unexpected internal error occurred.

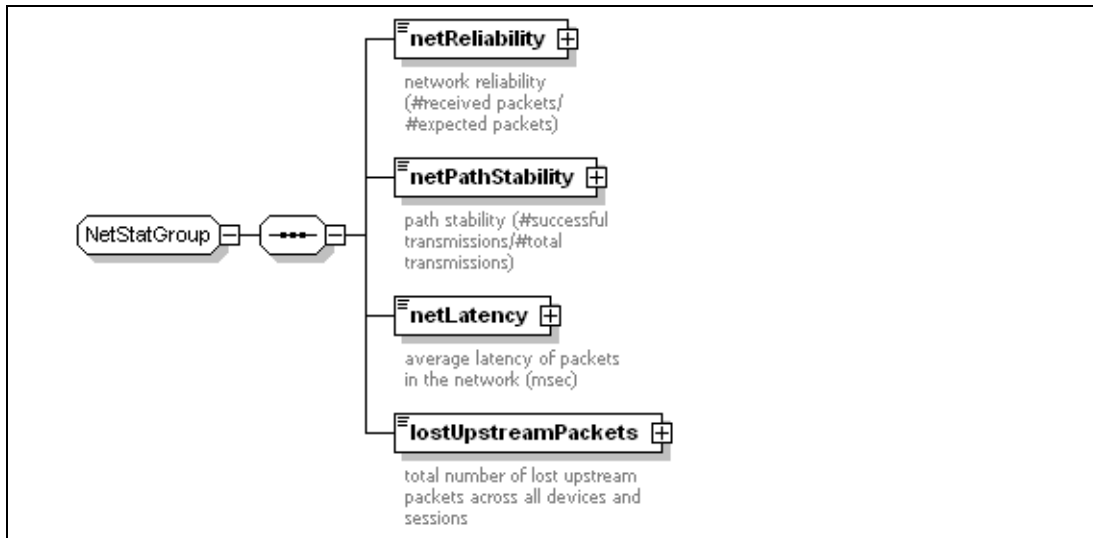
Finished indicates the OTAP was completed on this mote.

element **config/Network/Statistics**

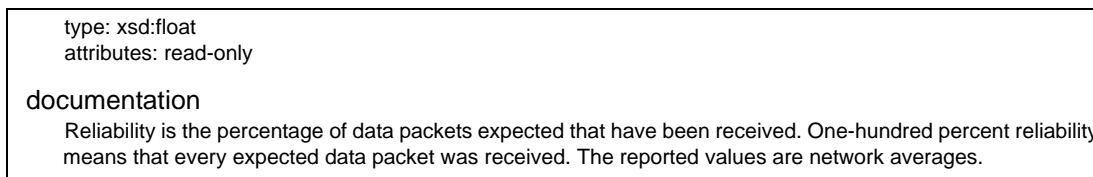
element **config/Network/Statistics/statCur**



element **NetStatGroup**



element **NetStatGroup/netReliability**



element NetStatGroup/netPathStability

type: xsd:float
 attributes: read-only

properties

minInclusive: 0
 maxInclusive: 100

documentation

Path stability is the percentage of transmitted packets that have successfully reached their destination over a given path. 100% path stability indicates reliable RF connectivity. The reported values are network averages.

element NetStatGroup/netLatency

type: xsd:unsignedLong
 attributes: read-only

properties

minInclusive: 0
 maxInclusive: 4294967295

documentation

Latency is the average time in milliseconds required for a data packet to travel from the originating mote to the manager. Latency may vary across the network. The value displayed represents the average network latency.

element NetStatGroup/lostUpstreamPackets

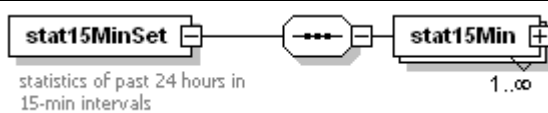
type: xsd:unsignedLong
 attributes: read-only

properties

minInclusive: 0
 maxInclusive: 4294967295

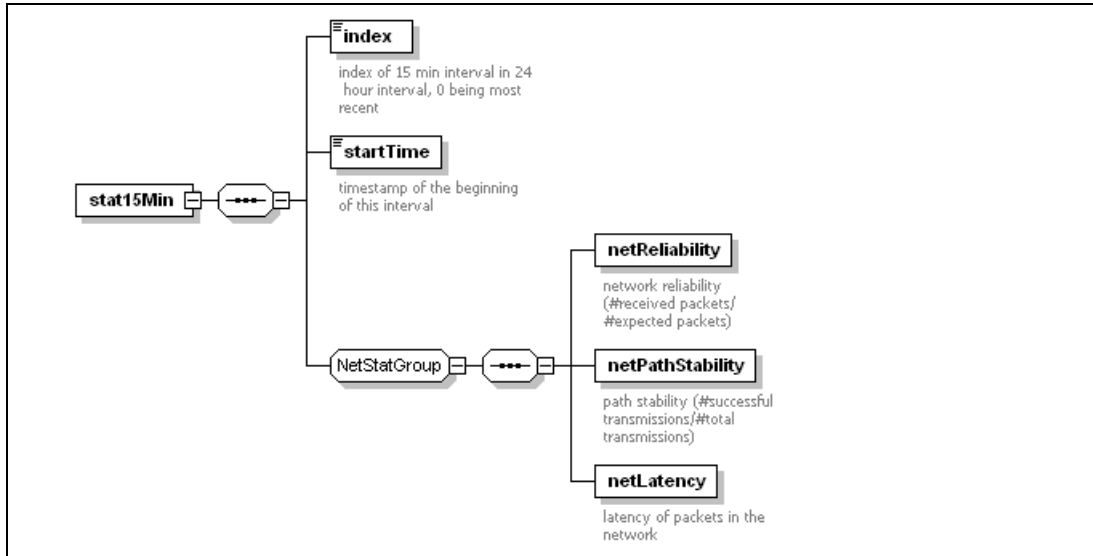
documentation

Total number of lost upstream packets across all devices and sessions. This field is only returned for config/Network/Statistics/Lifetime.

element config/Network/Statistics/stat15MinSet**documentation**

Statistics of past 24 hours in 15-minute intervals.

element **config/Network/Statistics/stat15MinSet/stat15Min**



element **config/Network/Statistics/stat15MinSet/stat15Min/index**

type: xsd:nonNegativeInteger
attributes: read-only

documentation
Index of 15-minute intervals in a 24-hour interval, 0 being most recent.

element **config/Network/Statistics/stat15MinSet/stat15Min/startTime**

type: dustTime
attributes: read-only

documentation
Timestamp of the beginning of this interval. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

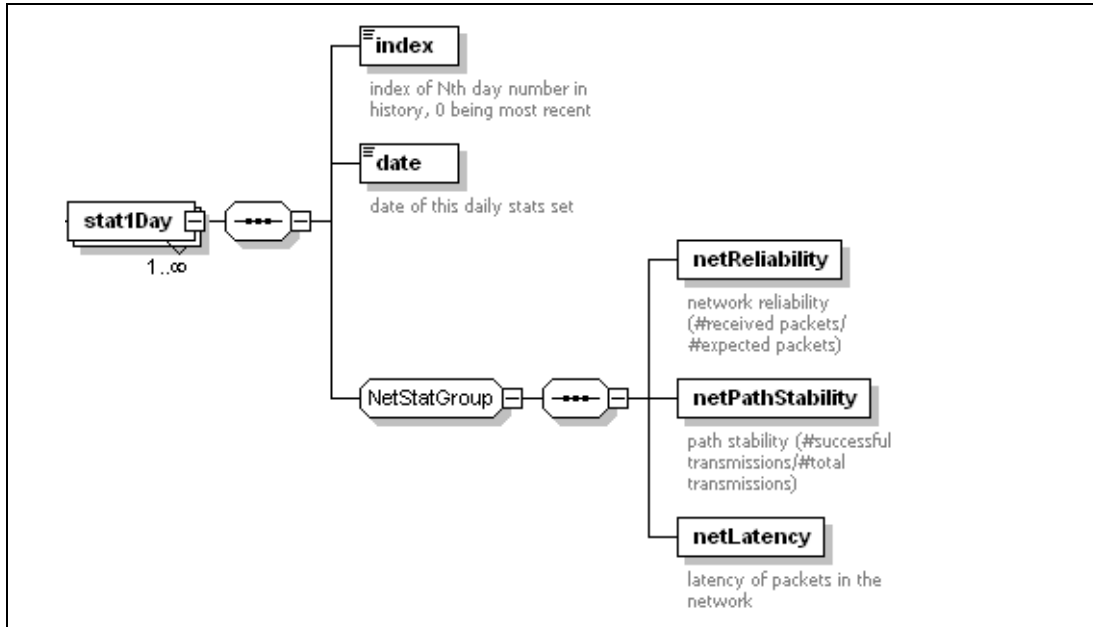
element **config/Network/Statistics/stat15MinSet/stat15Min/NetStatGroup**

See [“element NetStatGroup” on page 81](#).

element **config/Network/Statistics/stat1DaySet**

documentation
Daily statistics.

element `config/Network/Statistics/stat1DaySet/stat1Day`



element `config/Network/Statistics/stat1DaySet/stat1Day/index`

type: `xsd:nonNegativeInteger`
 attributes: read-only

documentation

Index of nth day number in history, 0 being most recent.

element `config/Network/Statistics/stat1DaySet/stat1Day/date`

type: `dustTime`
 attributes: read-only

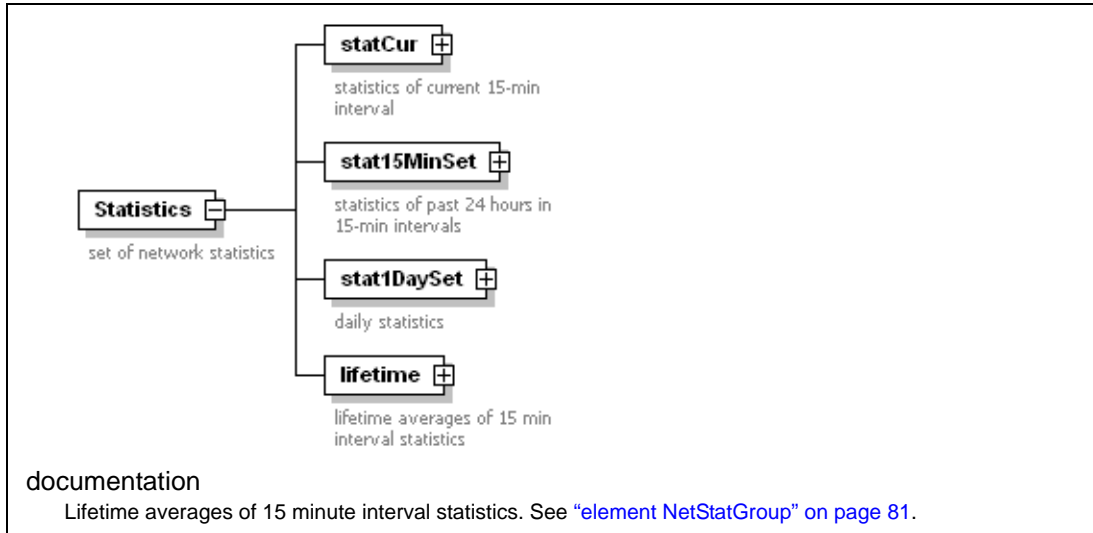
documentation

Date of this daily statistics set. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

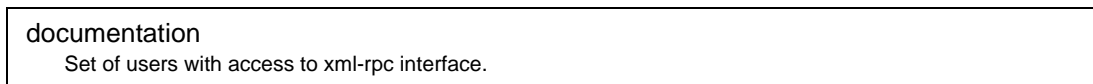
element `config/Network/Statistics/stat1DaySet/stat1Day/NetStatGroup`

See ["element NetStatGroup"](#) on page 81.

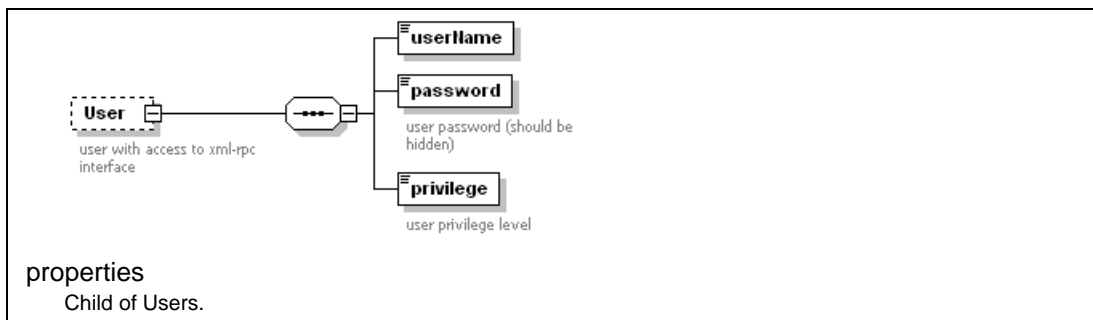
element **config/Network/Statistics/lifetime**



element **config/Users**



element **config/Users/User**



element **config/Users/User/userName**



element config/Users/User/password

type: xsd:string
 attributes: read/write

properties

minLength: 0
 maxLength: 16

documentation

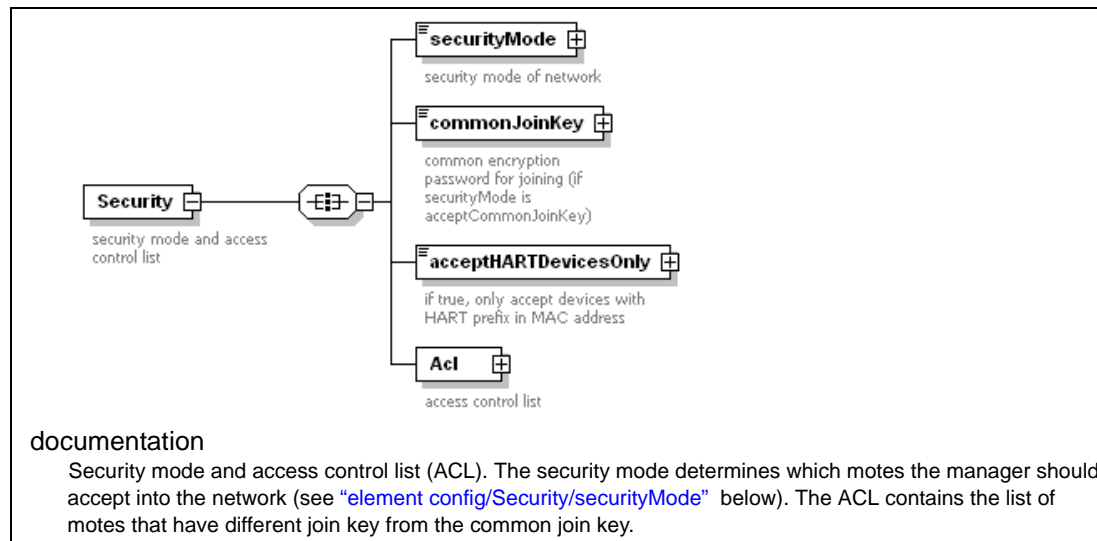
User password (should be hidden).

element config/Users/User/privilege

type: xsd:string
 attributes: read/write

documentation

User privilege level. Reserved for future use.

element config/Security

element `config/Security/securityMode`

type: xsd:string
 attributes: read/write

properties
 enumeration: acceptACL
 enumeration: acceptCommonJoinKey

documentation
 Security mode of network. The security mode determines which motes the manager should accept into the network. When the security mode is set to accept ACL, only motes on the access control list (ACL) may be accepted into the network. When the security mode set to accept the common join key, motes may be accepted into the network if they have the common join key or if they are listed on the ACL. After changing the security mode to accept ACL it is advised that you reset the network to remove any motes that may have previously joined the network using the common join key.
 When the security mode is changed, the manager issues a `sysConfigChange` event notification (see ["element notifications/event/sysConfigChange"](#) on page 121).

element `config/Security/commonJoinKey`

type: xsd:string
 attributes: read/write
 length: 32

documentation
 Network encryption password for joining. This is the common join key.

element `config/Security/acceptHARTDevicesOnly`

type: xsd:boolean
 attributes: read/write

documentation
 If true, the network only accepts devices with a HART prefix in the MAC address. address. By default this field is set to accept all devices (false).

element `config/Security/Acl`

documentation
 Access control list (ACL) is the list of motes that may be accepted into the network with their join keys. Motes on the ACL do not use the common join key. When you delete a mote from the ACL list the mote will not be able to rejoin the network the next time the mote resets.

element config/Security/Acl/Device/macAddress

type: xsd:string
attributes: read/write

documentation

MAC address of mote to be accepted into the network when the security mode is set to accept ACL. For information about security modes, see “element config/Security/securityMode” on page 87.

element config/Security/Acl/Device/joinKey

type: xsd:string
attributes: read/write

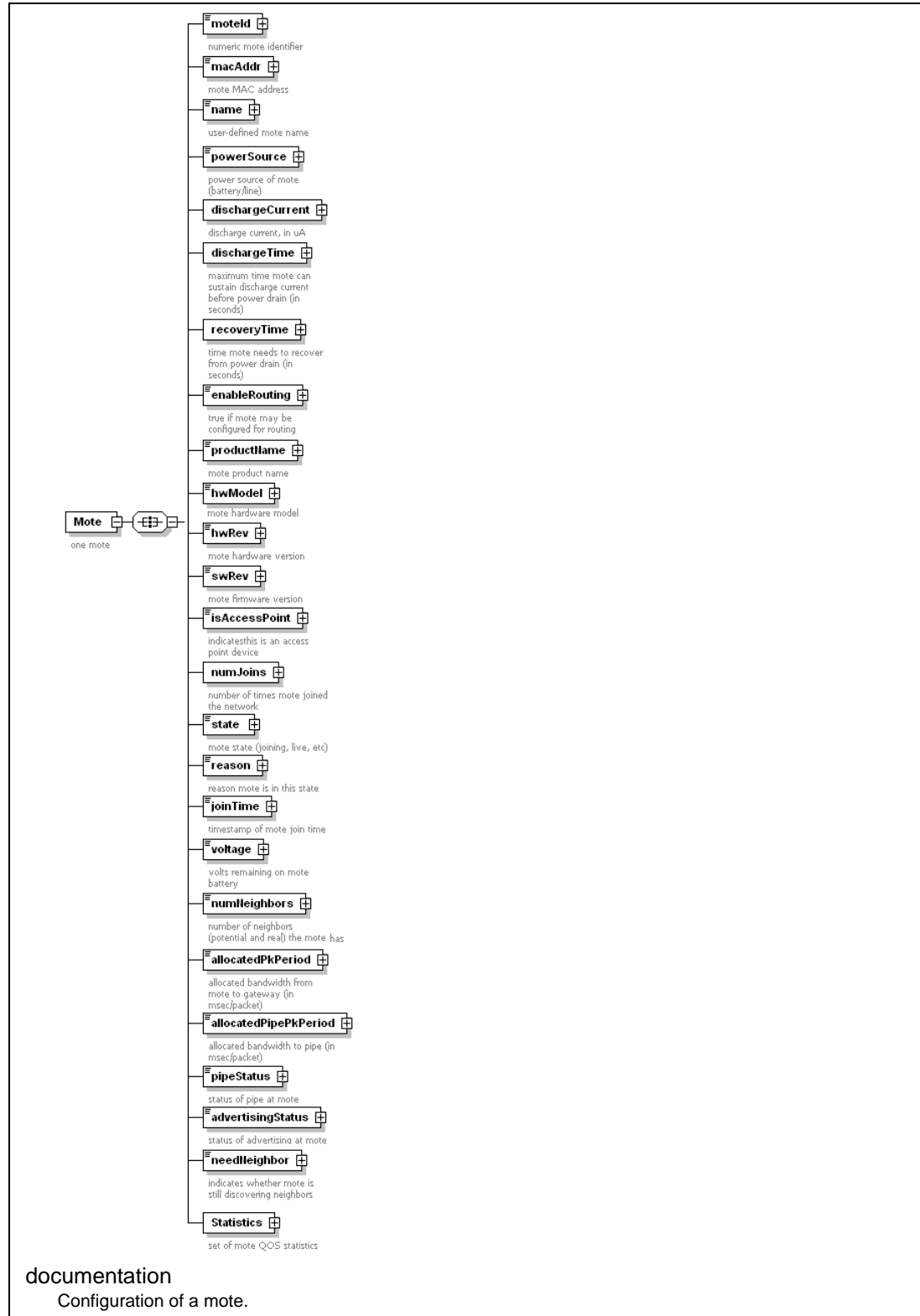
properties

length: 32

documentation

Join key of mote to be accepted into the network.

element **config/Motes**



element config/Motes (cont)

```

example
<Motes>
  <Mote>
    <moteId>18</moteId>
    <macAddr>00-00-00-00-00-00-19-04</macAddr>
    <name>mote18</name>
    <powerSource>AA_L91</powerSource>
    <enableRouting>>true</enableRouting>
    <hwModel>A0000</hwModel>
    <hwRev>A0</hwRev>
    <swRev>1.1-100</swRev>
    <isAccessPoint>>false</isAccessPoint>
    <numJoins>3</numJoins>
    <state>live</state>
    <reason/>
    <joinTime>100009231</joinTime>
    <voltage>3.0</voltage>
  </Mote>
</Motes>

```

element config/Motes/Mote/moteld

type: xsd:unsignedShort
 attributes: read/write

properties
 minInclusive: 1
 maxInclusive: 528

documentation

Numeric mote identifier.

Note that you cannot delete a mote if it is still in the network. The mote must first be decommissioned (see “decommissionDevice” on page 37) and then physically removed from the network. The manager maintains a list of motes that are in the network or are expected to be in the network. Deleting a mote removes the mote from manager’s list and from the ACL. To delete a mote, use the command “deleteConfig” on page 38 and specify <config><Motes><Mote><macAddress> as the document request.

element config/Motes/Mote/macAddr

type: xsd:string
 attributes: read/write

documentation

Mote MAC address. For information on deleting a mote, see “element config/Motes/Mote/moteld” on page 90.

element config/Motes/Mote/name

type: xsd:string
attributes: read/write

properties

minLength: 0
maxLength: 16

element config/Motes/Mote/powerSource

type: xsd:string
attributes: read/write

properties

enumeration: line
enumeration: battery
enumeration: rechargeable/power scavenging

documentation

Mote power source (line, battery, or rechargeable/power scavenging).

element config/Motes/Mote/dischargeCurrent

type: xsd:unsignedShort
attributes: read-only

properties

minInclusive: 0
maxInclusive: 65535

documentation

The discharge current indicates the discharge current, in μ A.
Note: The discharge current, discharge time, and recovery time are collectively used to describe the current sourcing capabilities of the three possible types of power supply feeding the mote.

element config/Motes/Mote/dischargeTime

type: xsd:unsignedLong
attributes: read-only

properties

minInclusive: 0
maxInclusive: 4294967295

documentation

The discharge time is the maximum time (in seconds) that the power supply can sustain the discharge current before experiencing a voltage drop.
Note: The discharge current, discharge time, and recovery time are collectively used to describe the current sourcing capabilities of the three possible types of power supply feeding the mote.

element config/Motes/Mote/recoveryTime

type: xsd:unsignedLong
attributes: read-only

properties

minInclusive: 0
maxInclusive: 4294967295

documentation

The recovery time is the time (in seconds) required by the power supply to recover from a power drain (for example, recharge its capacitors).

Note: The discharge current, discharge time, and recovery time are collectively used to describe the current sourcing capabilities of the three possible types of power supply feeding the mote.

element config/Motes/Mote/enableRouting

type: xsd:boolean
attributes: read/write

documentation

“True” if mote may be configured for routing.

element config/Motes/Mote/productName

type: xsd:string
attributes: read-only

properties

minLength: 0
maxLength: 16

documentation

Mote product name.

element config/Motes/Mote/hwModel

type: xsd:string
attributes: read-only

properties

minLength: 0
maxLength: 16

documentation

Mote hardware model.

element config/Motes/Mote/hwRev

type: restriction of xsd:string attributes: read-only
properties minLength: 0 maxLength: 16
documentation Mote hardware version.

element config/Motes/Mote/swRev

type: xsd:string attributes: read-only
properties minLength: 0 maxLength: 16
documentation Mote firmware version. During mote startup, the version number is initially populated with the placeholder value "0.0.0-0" until the mote becomes operational and reports its value through the network (this may take a few minutes).

element config/Motes/Mote/isAccessPoint

type: xsd:boolean attributes: read-only
documentation Indicates that this is an access point device.

element config/Motes/Mote/numJoins

type: xsd:unsignedShort attributes: read-only
properties minInclusive: 0 maxInclusive: 65535
documentation Total number of times mote joined the network.

element config/Motes/Mote/state

type: xsd:string
attributes: read-only

properties

enumeration: Idle
enumeration: Lost
enumeration: Negotiating1
enumeration: Negotiating2
enumeration: Connected
enumeration: Operational
enumeration: Disconnecting

documentation

Mote state (idle, lost, negotiating1, negotiating2, connected, operational, disconnecting).

element config/Motes/Mote/reason

type: xsd:string

properties

enumeration: MAXMOTES (maximum number of motes was reached)
enumeration: UNREACH (unreachable)
enumeration: NOTCONN (not connected)
enumeration: CFGERR (configuration error)

documentation

Indicates why the mote is in current state.

element config/Motes/Mote/joinTime

type: dustTime

documentation

Timestamp of mote join time. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element config/Motes/Mote/voltage

type: xsd:float
attributes: read-only

documentation

The voltage is the reading from the last measurement of the mote power supply.

element config/Motes/Mote/numNeighbors

type: xsd:unsignedShort attributes: read-only
properties minInclusive: 0 maxInclusive 65535
documentation The number of neighbors (potential and connected) that the mote has.

element config/Motes/Mote/allocatedPkPeriod

type: xsd:unsignedLong attributes: read-only
properties minInclusive: 0 maxInclusive: 4294967295
documentation Currently allocated bandwidth (in msec/packet) from mote to gateway.

element config/Motes/Mote/allocatedPipePkPeriod

type: xsd:unsignedLong attributes: read-only
properties minInclusive: 0 maxInclusive: 4294967295
documentation Currently allocated bandwidth (in msec/packet) for the pipe.

element config/Motes/Mote/pipeStatus

type: xsd:string attributes: read-only
properties enumeration: off enumeration: pending enumeration: on_bi (pipe is on for bidirectional traffic) enumeration: on_up (pipe is on for upstream traffic) enumeration: on_down (pipe is on for downstream traffic)
documentation Status of pipe at mote. The status "pending" means the manager is in the process of changing the state of the pipe from on to off (or vice-versa).

element config/Motes/Mote/advertisingStatus

type: xsd:string
attributes: read-only

properties

enumeration: on
enumeration: off
enumeration: pending

documentation

Status of advertising at mote.

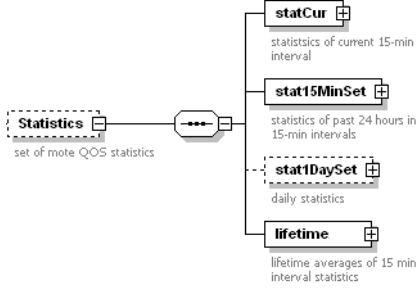
element config/Motes/Mote/needNeighbor

type: xsd:boolean
attributes: read-only

documentation

Indicates whether a mote has only one parent and (or) insufficient links. A well formed network should have only one mote (the "single-parent" mote in the first hop) for which needNeighbor is true. If this parameter is true for any other mote, it indicates that an additional mote needs to be added nearby.

element **config/Motes/Mote/Statistics**

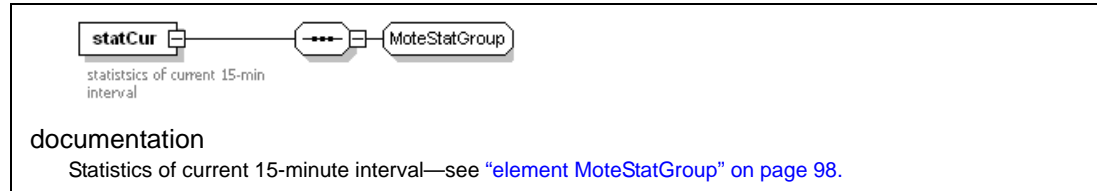


example

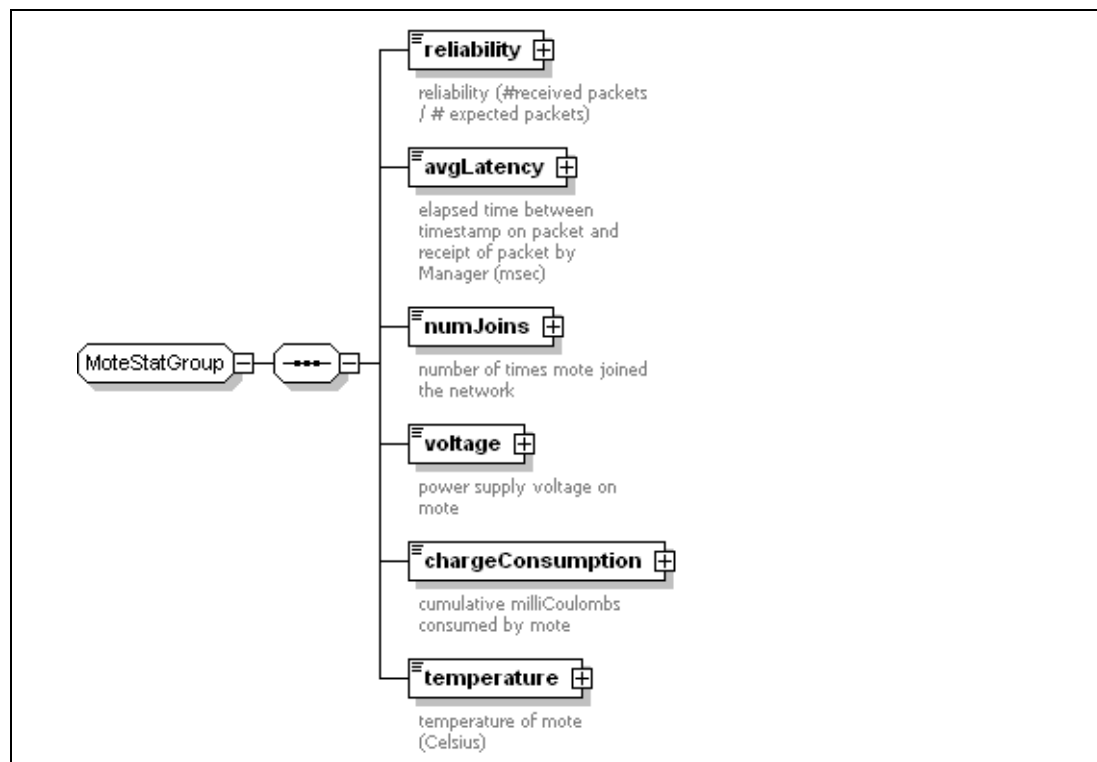
```

<Motes>
  <Mote>
    <moteId>18</moteId>
    <macAddr>00-00-00-00-00-00-19-04</macAddr>
    <Statistics>
      <lifetime>
        <reliability>99.99</reliability>
        <avgLatency>10.2</avgLatency>
        <numJoins>3</numJoins>
        <voltage>3.0</voltage>
        <chargeConsumption>3.0</chargeConsumption>
        <temperature>25</temperature>
      </lifetime>
    </Statistics>
  </Mote>
</Motes>
    
```

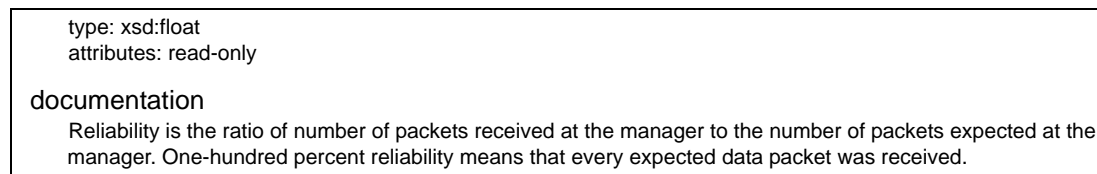
element `config/Motes/Mote/Statistics/statCur`



element `MoteStatGroup`



element `MoteStatGroup/reliability`



element MoteStatGroup/avgLatency

type: xsd:unsignedLong
 attributes: read-only

documentation
 Elapsed time between timestamp on packet and receipt of packet by the manager.

element MoteStatGroup/numJoins

type: xsd:unsignedShort
 attributes: read-only

properties
 minInclusive: 0
 maxInclusive 65535

documentation
 Number of times mote joined the network in this statistic period.

element MoteStatGroup/voltage

type: xsd:float
 attributes: read-only

documentation
 Volts remaining on mote battery.

element MoteStatGroup/chargeConsumption

type: xsd:nonNegativeInteger
 attributes: read-only

documentation
 Cumulative millicoulombs consumed by mote. Charge calculations are based on nominal part performance.

element MoteStatGroup/temperature

type: xsd:float
 attributes: read-only

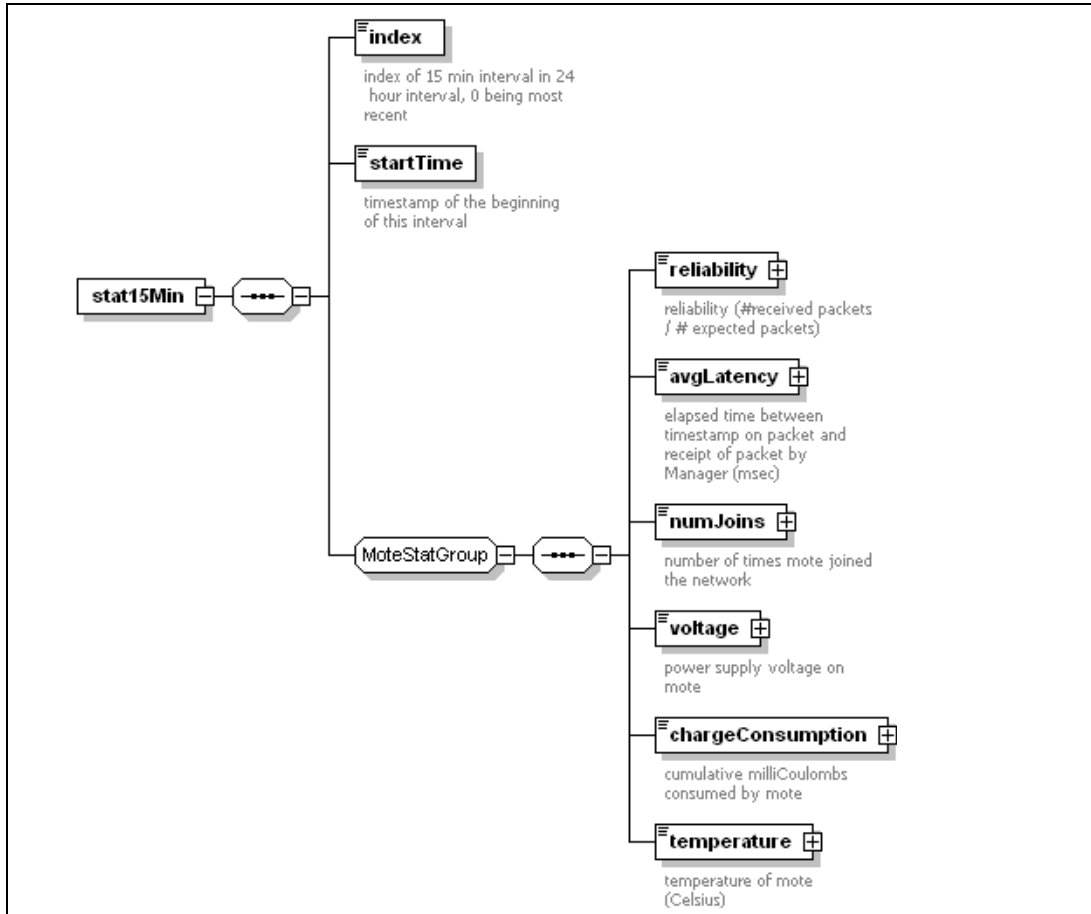
documentation
 Temperature of mote (Celsius).

element config/Motes/Mote/Statistics/stat15MinSet

statistics of past 24 hours in 15-min intervals

1..∞

element `config/Motes/Mote/Statistics/stat15MinSet/stat15Min`



element `config/Motes/Mote/Statistics/stat15MinSet/stat15Min/index`

type: `xsd:nonNegativeInteger`
attributes: read-only

documentation

Index of 15 minute interval in 24 hour interval, 0 being most recent.

element `config/Motes/Mote/Statistics/stat15MinSet/stat15Min/startTime`

type: `dustTime`
attributes: read-only

documentation

Timestamp of the beginning of this interval. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element `config/Motes/Mote/Statistics1stat15MinSet/stat15Min/MoteStatGroup`

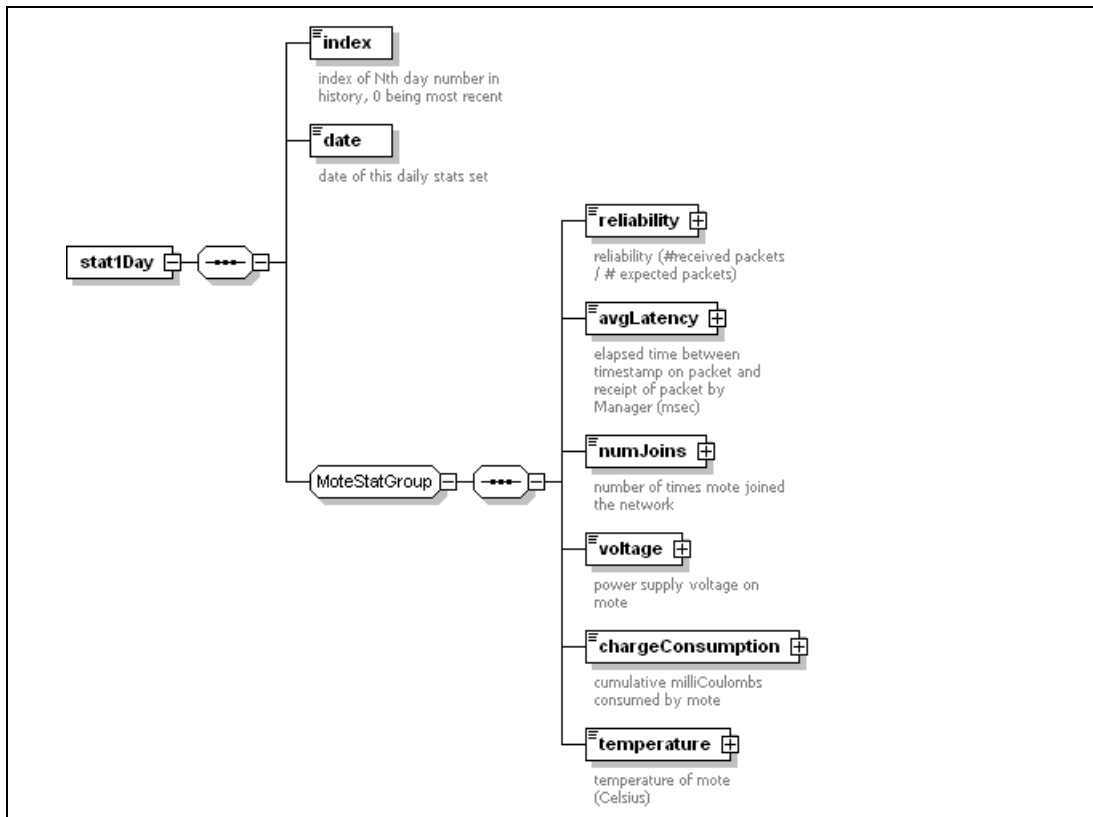
documentation

See [“element MoteStatGroup” on page 98](#).

element **config/Motes/Mote/Statistics/stat1DaySet**



element **config/Motes/Mote/Statistics/stat1DaySet/stat1Day**



element **config/Motes/Mote/Statistics/stat1DaySet/stat1Day/index**

type: xsd:nonNegativeInteger
 attributes: read-only

documentation
 Index of nth day number in history, 0 being most recent.

element `config/Motes/Mote/Statistics/stat1DaySet/stat1Day/date`

type: dustTime
attributes: read-only

documentation

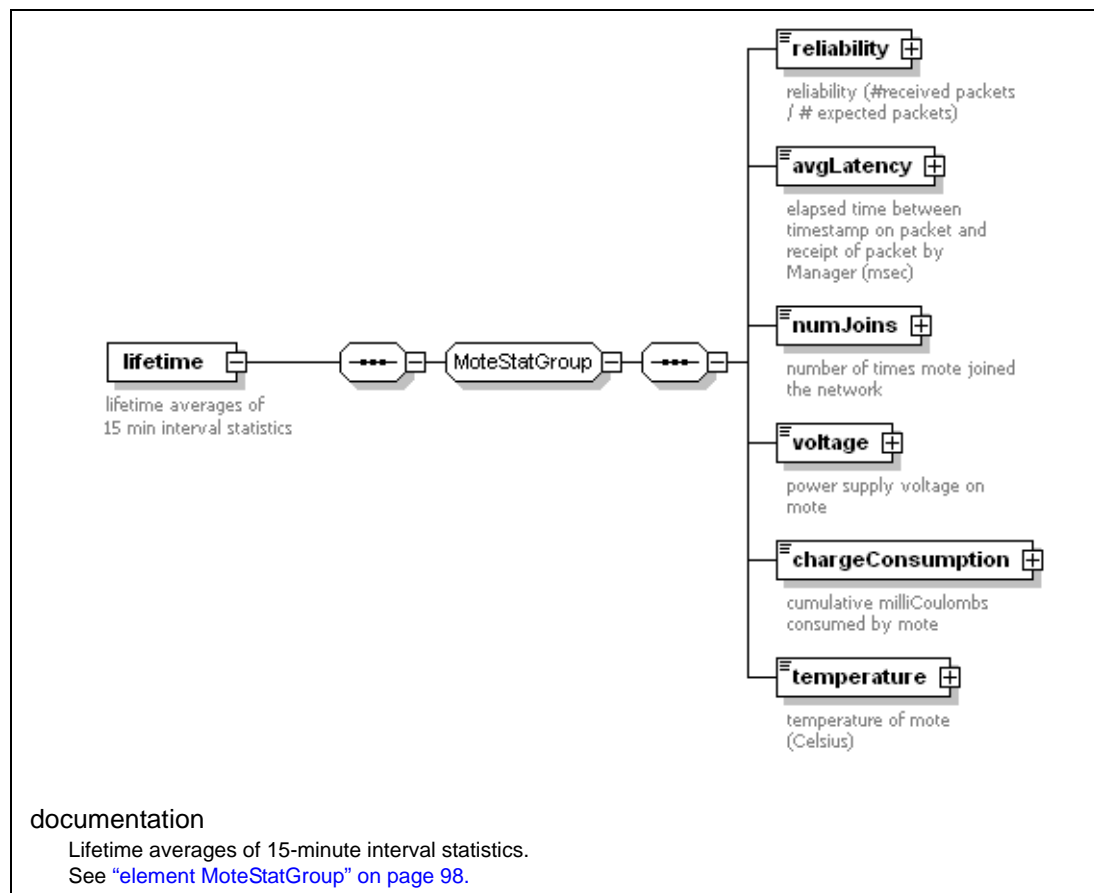
Date of this daily statistics set. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element `config/Motes/Mote/Statistics/stat1DaySet/stat1Day/MoteStatGroup`

documentation

See [“element MoteStatGroup” on page 98](#).

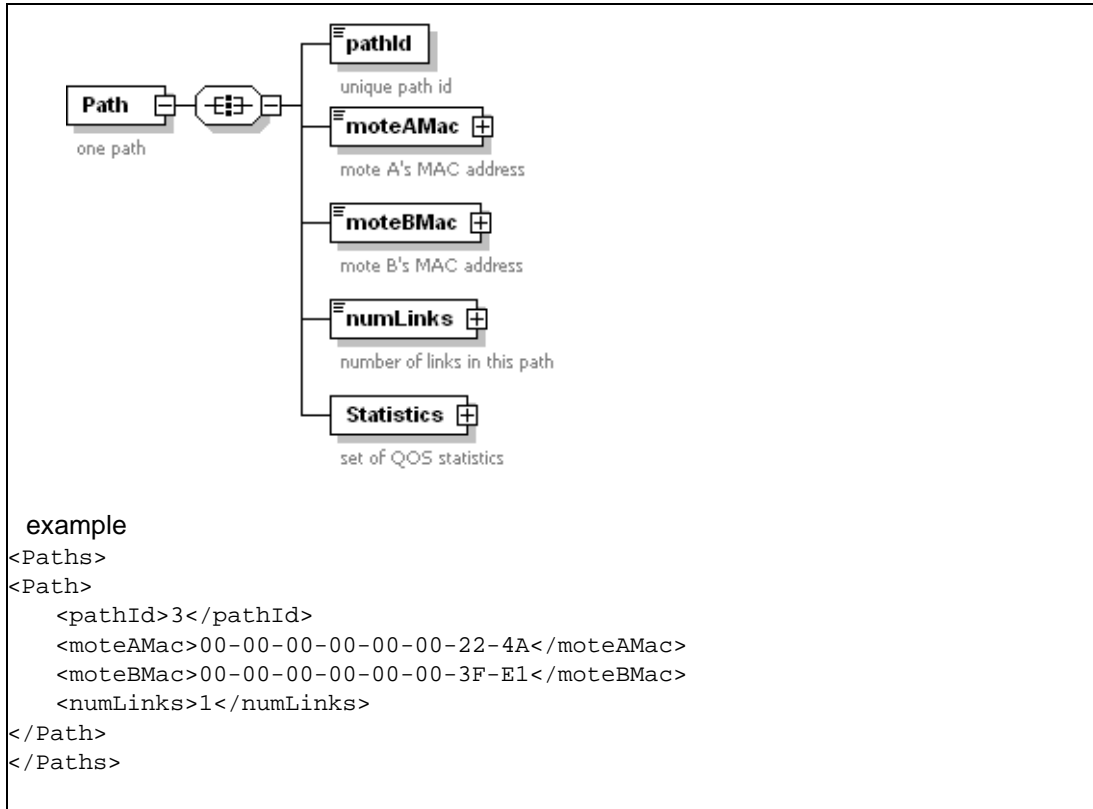
element `config/Motes/Mote/Statistics/lifetime`



documentation

Lifetime averages of 15-minute interval statistics.
See [“element MoteStatGroup” on page 98](#).

element **config/Paths**



element **config/Paths/Path/pathId**

type: `xsd:nonNegativeInteger`
 attributes: read-only

documentation
 Unique path ID.

element **config/Paths/Path/moteAMac**

type: `xsd:string`
 attributes: read-only

documentation
 Mote A's MAC address.

element config/Paths/Path/moteBMac

type: xsd:string
 attributes: read-only

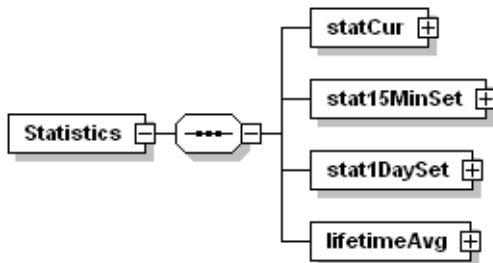
documentation
 Mote B's MAC address.

element config/Paths/Path/numLinks

type: xsd:unsignedShort
 attributes: read-only

properties
 minInclusive: 0
 maxInclusive: 65535

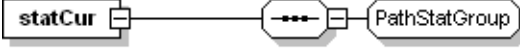
documentation
 Number of links in this path.

element config/Paths/Path/Statistics**example**

```

<Paths>
  <Path>
    <pathId>45</pathId>
    <Statistics>
      <lifetime>
        <abPwr>-71</abPwr>
        <baPwr>-72</baPwr>
        <stability>60.3</stability>
      </lifetime>
    </Statistics>
  </Path>
</Paths>
  
```

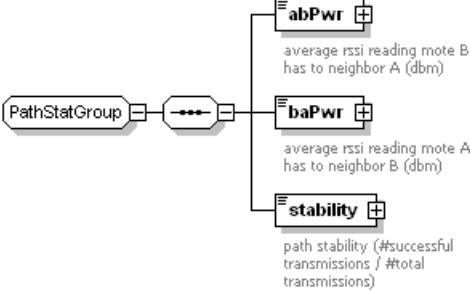
element **config/Paths/Path/Statistics/statCur**



statistics of current 15-min interval

documentation
 Statistics of current 15-minute interval.
 See "element PathStatGroup" on page 105.

element **PathStatGroup**



average rssi reading mote B has to neighbor A (dbm)

average rssi reading mote A has to neighbor B (dbm)

path stability (#successful transmissions / #total transmissions)

element **PathStatGroup/abPwr**

type: xsd:short
 attributes: read-only

properties
 minInclusive: -128
 maxInclusive: 127

documentation
 Average RSSI reading mote B has to neighbor mote A (dBm).

element **PathStatGroup/baPwr**

type: xsd:short
 attributes: read-only

properties
 minInclusive: -128
 maxInclusive: 127

documentation
 Average RSSI reading mote A has to neighbor mote B (dBm).

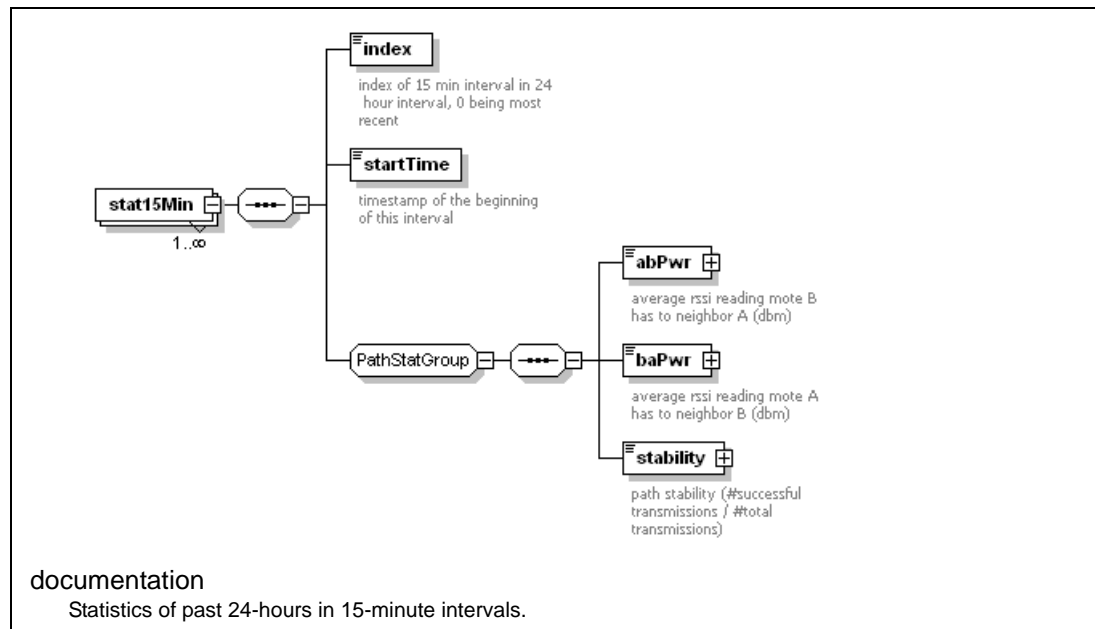
element `PathStatGroup/stability`

type: xsd:float
attributes: read-only

documentation

Path stability (number of successful transmissions divided by total number of transmissions).

element `config/Paths/Path/Statistics/stat15MinSet/stat15Min`



element `config/Paths/Path/Statistics/stat15MinSet/stat15Min/index`

type: xsd:nonNegativeInteger
attributes: read-only

documentation

Index of 15-minute interval in a 24-hour interval, 0 being most recent.

element `config/Paths/Path/Statistics/stat15MinSet/stat15Min/startTime`

type: dustTime
attributes: read-only

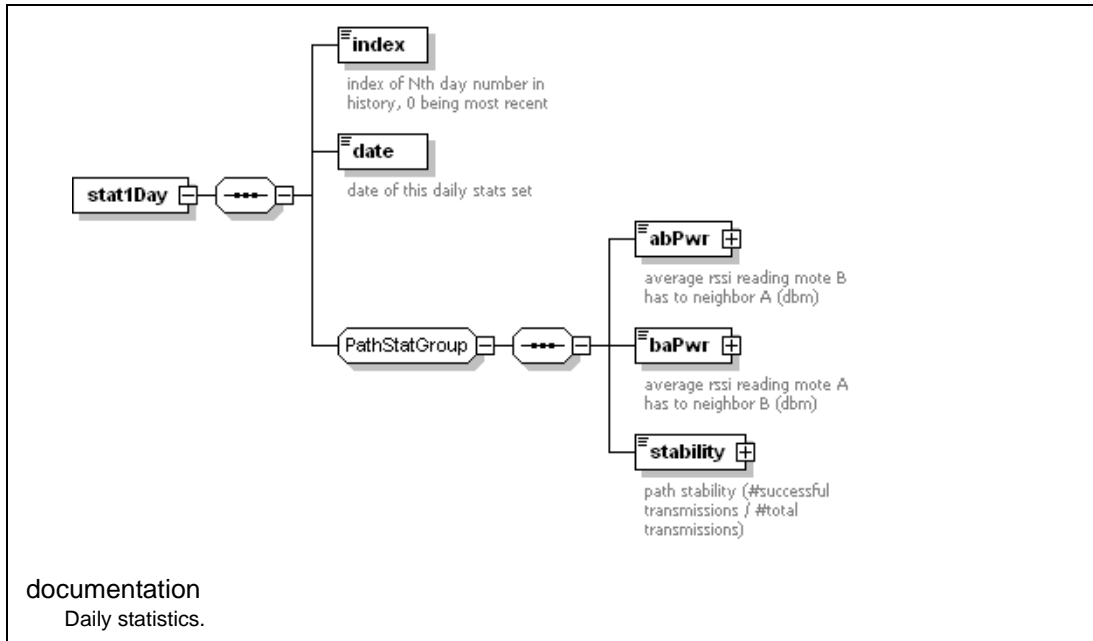
documentation

Timestamp of the beginning of this interval. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element **config/Paths/Path/Statistics/stat15MinSet/stat15Min/PathStatGroup**

See "element PathStatGroup" on page 105.

element **config/Paths/Path/Statistics/stat1DaySet/stat1Day**



element **config/Paths/Path/Statistics/stat1DaySet/stat1Day/index**

type: xsd:nonNegativeInteger
attributes: read-only

documentation
Index of nth day number in history, 0 being most recent.

element **config/Paths/Path/Statistics/stat1DaySet/stat1Day/date**

type: dustTime
attributes: read-only

documentation
Date of this daily statistics set. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element **config/Paths/Path/Statistics/stat1DaySet/stat1Day/PathStatGroup**

See "element PathStatGroup" on page 105.

element **config/Paths/Path/Statistics/lifetime**

lifetime
lifetime averages of 15 min interval statistics

type: xsd:nonNegativeInteger

documentation
Lifetime averages of 15-minute interval statistics.
See "element PathStatGroup" on page 105.

element **config/Alarms**

Alarm
network alarm

alarmType

- eventInfo**
 - timeStamp**
time event was generated
 - eventId**
numeric identifier
- alarmInfo**
 - moteDown**
mote has lost communication
 - slaReliability**
network reliability does not meet Service Level Agreement
 - slaLatency**
network latency does not meet Service Level Agreement
 - slaStability**
network stability does not meet Service Level Agreement
 - maxMotesReached**
max motes was reached

type: alarmType

documentation
Network alarms.

element config/Alarms/Alarm/timeStamp

type: dustTime

documentation

Time the event was generated. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element config/Alarms/Alarm/eventId

type: xsd:nonNegativeInteger

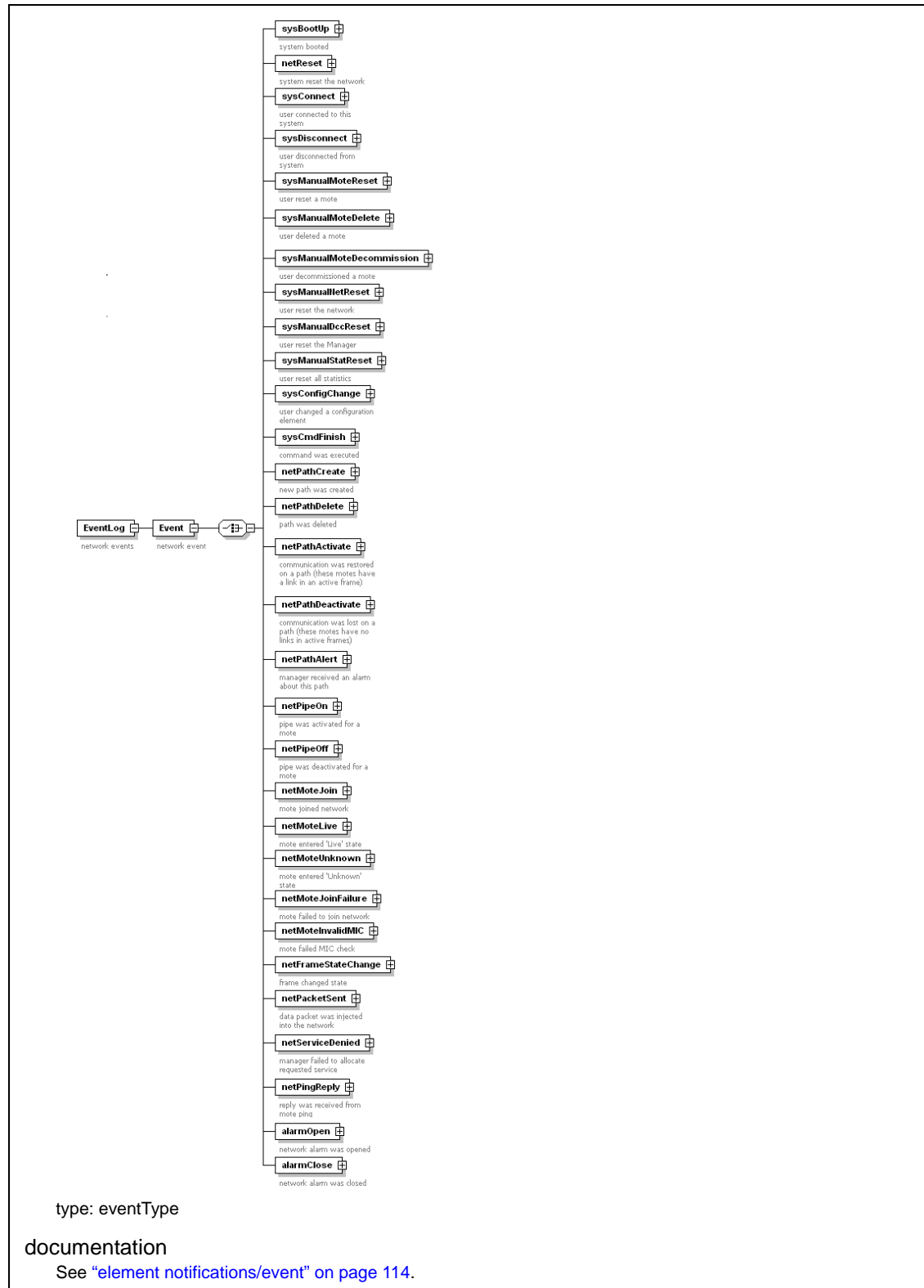
documentation

Numeric identifier.

element config/Alarms/Alarm/alarmInfo**documentation**

For alarmInfo, see ["complexType alarmInfo" on page 141](#).

element config/EventLog

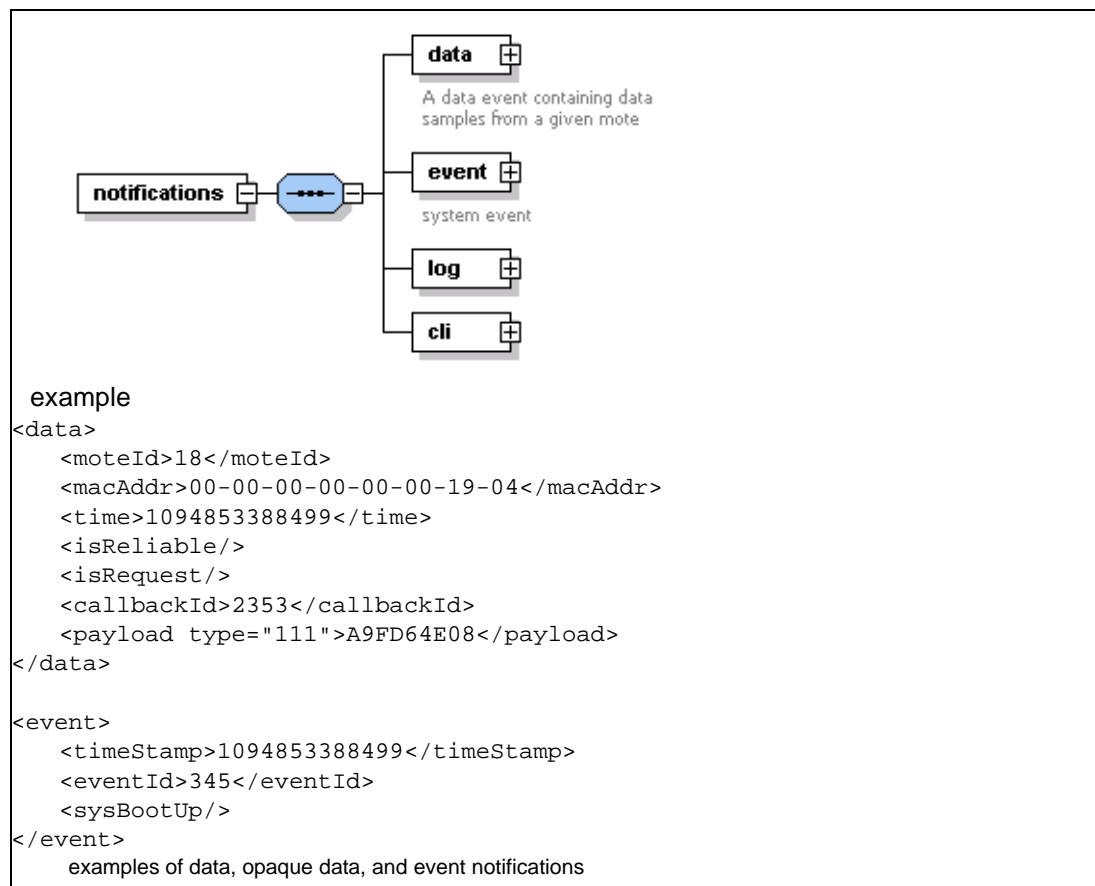


Notification Schema Reference

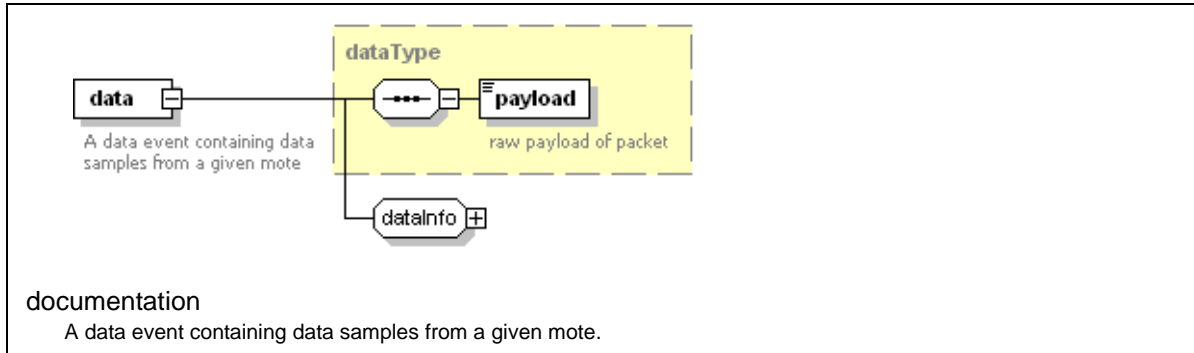
This section documents the XML schema that describes the format of the notification messages that a SmartMesh manager sends to client applications. Subscribed client applications listen on the notification channel to process the XML-wrapped stream of information about data and events in the network.

- [“element notifications”](#)
- [“element notifications/data”](#)
- [“element notifications/event”](#)
- [“element notifications/log”](#)
- [“element notifications/cli”](#)

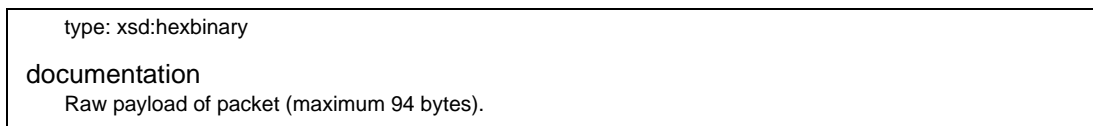
element notifications



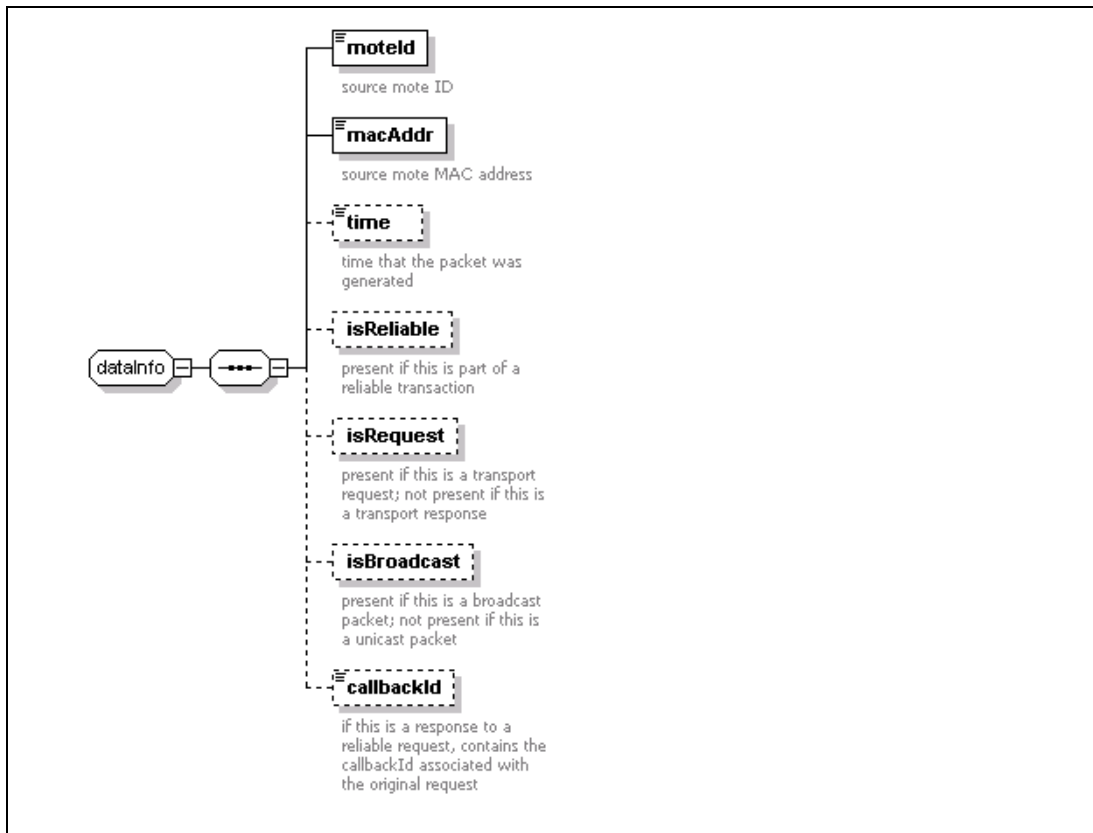
element **notifications/data**



element **notifications/data/payload**



element **notifications/data/dataInfo**



element notifications/data/dataInfo/moteld

type: xsd:positiveInteger
properties minInclusive: 1 maxInclusive: 528
documentation Source mote ID.

element notifications/data/dataInfo/macAddr

type: xsd:string
documentation Source mote MAC address.

element notifications/data/dataInfo/time

type: dustTime
documentation Time that the packet was generated. The time is represented as the number of milliseconds since 00:00:00:000 1/1/1970 GMT.

element notifications/data/dataInfo/isReliable

type: None
documentation Present if this is part of a reliable (acknowledged) transaction. If not present, the notification is a best-effort communication.

element notifications/data/dataInfo/isRequest

type: None
documentation Present if this is a transport request. If not present, this is a transport response.

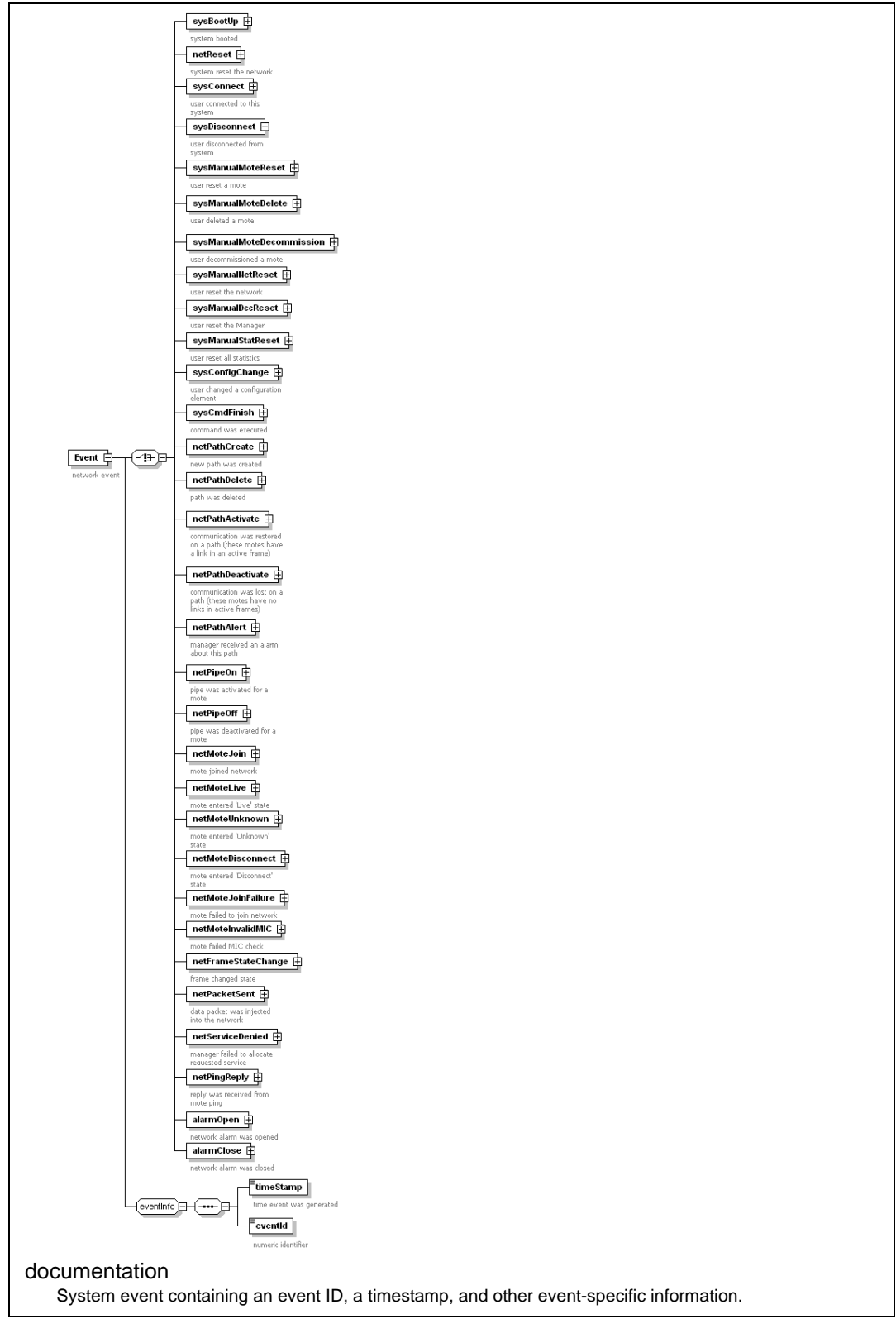
element notifications/data/dataInfo/isBroadcast

type: None
documentation Present if this is a broadcast response. If not present, this is a unicast response.

element notifications/data/dataInfo/callbackId

type: xsd: integer
documentation If this data notification is a response to a reliable request, this is the callbackId associated with the original request. If this data notification is a reliable request from a field device, the callbackId must be used in the sendResponse command to return a response to the request.

element notifications/event



documentation

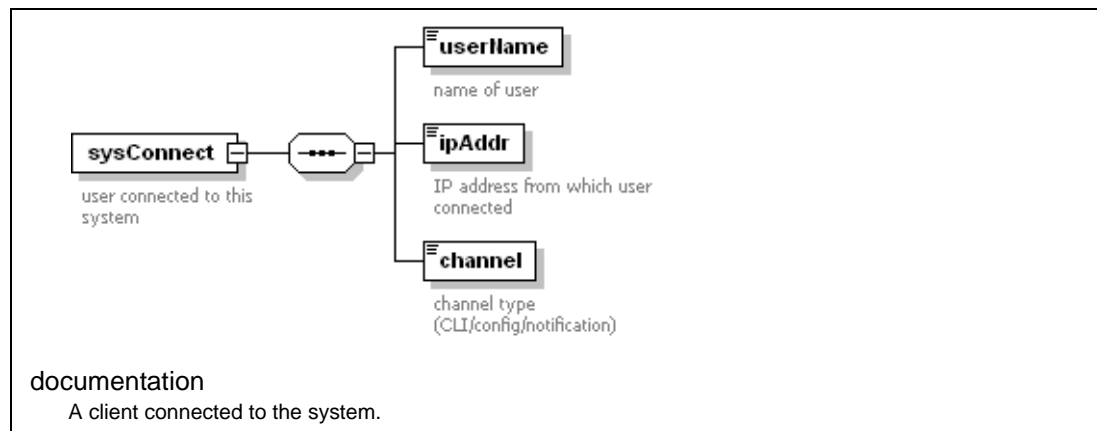
System event containing an event ID, a timestamp, and other event-specific information.

element notifications/event/sysBootUp

documentation
System started up.

element notifications/event/netReset

documentation
The system reset the network.

element notifications/event/sysConnect**element notifications/event/sysConnect/userName**

type: xsd:string

properties

minLength: 0
maxLength: 16

documentation

Username used to connect to the system. Internal notifications are indicated by the username "rpc-user."

element notifications/event/sysConnect/ipAddr

type: xsd:string

documentation

IP address from which user connected to the system.

element notifications/event/sysConnect/channel

<p>type: xsd:string</p> <p>properties</p> <ul style="list-style-type: none"> enumeration: CLI enumeration: config enumeration: notification <p>documentation</p> <p>Channel type used to connect to system.</p>
--

element notifications/event/sysDisconnect

<p>documentation</p> <p>User disconnected from the system. Internal notifications are indicated by the username "rpc-user."</p>
--

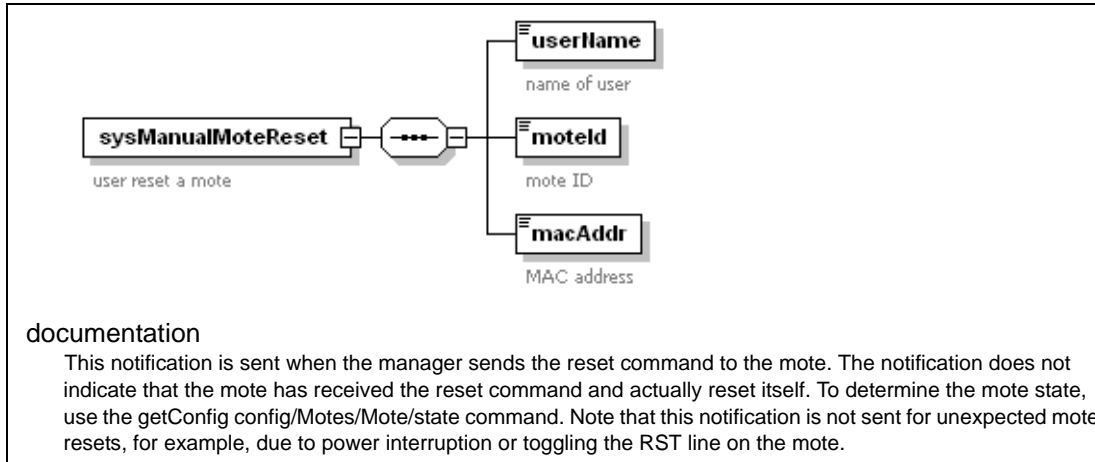
element notifications/event/sysDisconnect/userName

<p>type: xsd:string</p> <p>properties</p> <ul style="list-style-type: none"> minLength: 0 maxLength: 16 <p>documentation</p> <p>Name of user who disconnected from the system.</p>
--

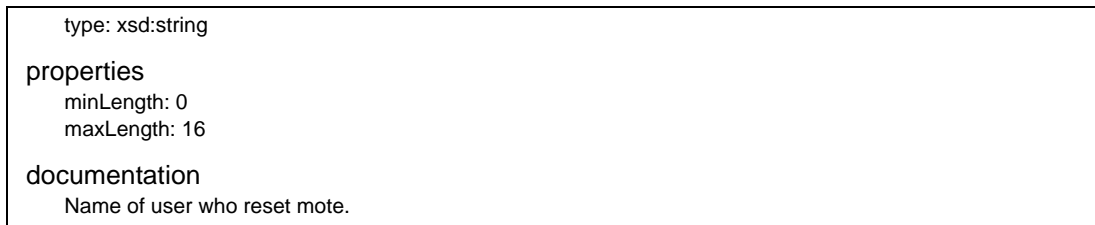
element notifications/event/sysDisconnect/channel

<p>type: xsd:string</p> <p>properties</p> <ul style="list-style-type: none"> enumeration: CLI enumeration: config enumeration: notification <p>documentation</p> <p>Channel type used to disconnect from system.</p>

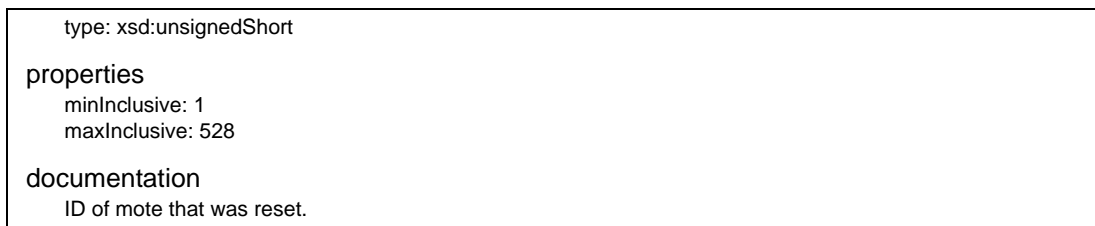
element `notifications/event/sysManualMoteReset`



element `notifications/event/sysManualMoteReset/userName`



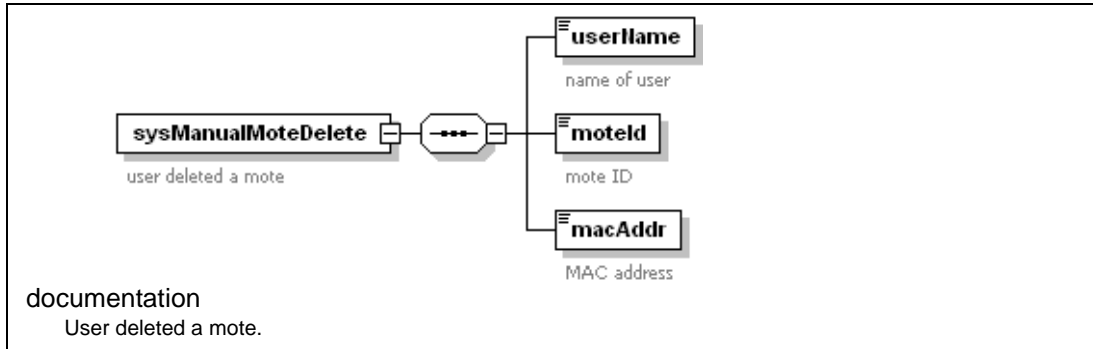
element `notifications/event/sysManualMoteReset/motelId`



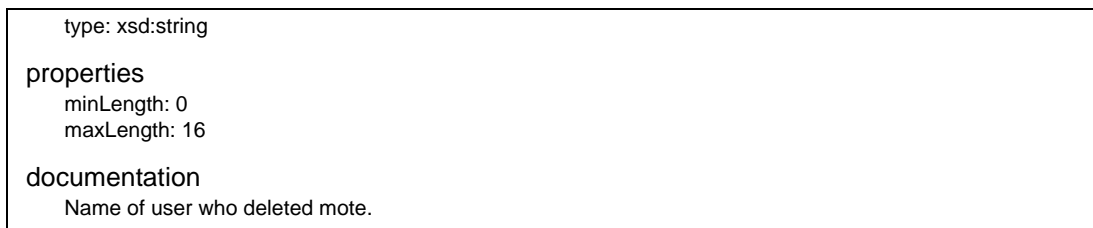
element `notifications/event/sysManualMoteReset/macAddr`



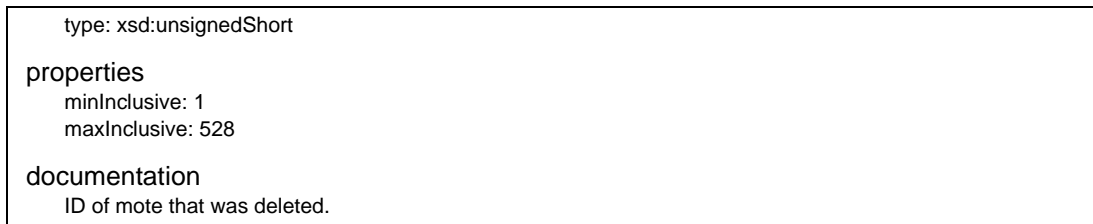
element `notifications/event/sysManualMoteDelete`



element `notifications/event/sysManualMoteDelete/userName`



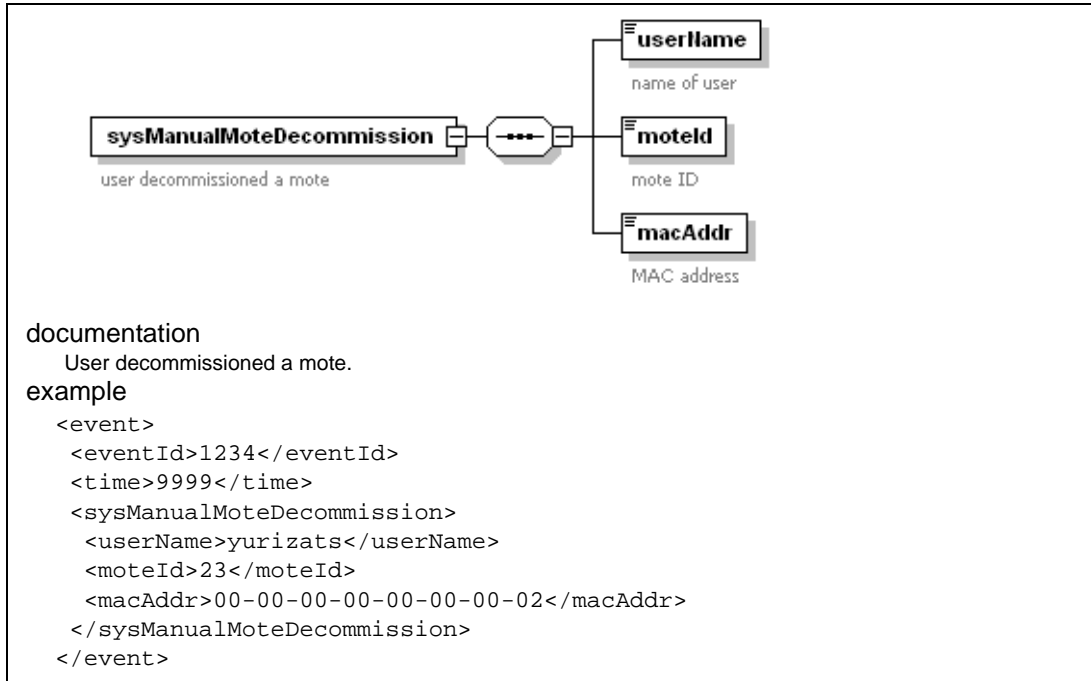
element `notifications/event/sysManualMoteDelete/motelId`



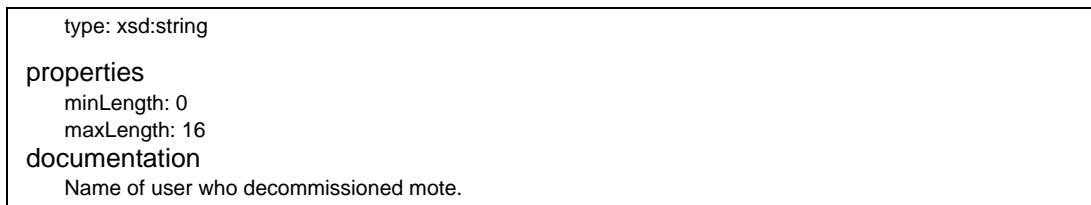
element `notifications/event/sysManualMoteDelete/macAddr`



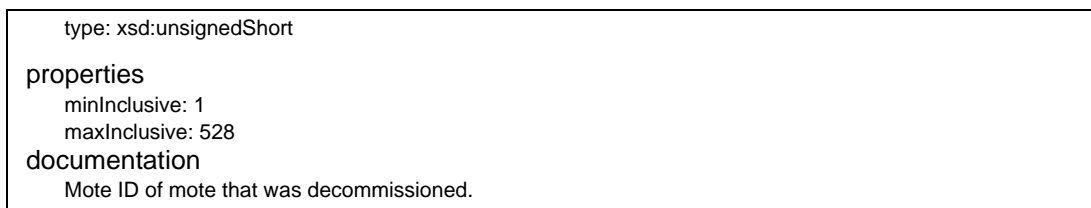
element `notifications/event/sysManualMoteDecommission`



element `notifications/event/sysManualMoteDecommission/userName`



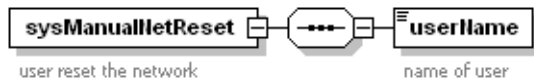
element `notifications/event/sysManualMoteDecommission/moteId`



element `notifications/event/sysManualMoteDecommission/macAddr`

type: xsd:string
 documentation
 MAC address of mote that was decommissioned.

element `notifications/event/sysManualNetReset`

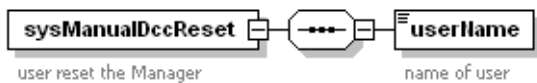


documentation
 User reset the network.

element `notifications/event/sysManualNetReset/username`

type: xsd:string
 properties
 minLength: 0
 maxLength: 16
 documentation
 Name of user who reset the network.

element `notifications/event/sysManualDccReset`

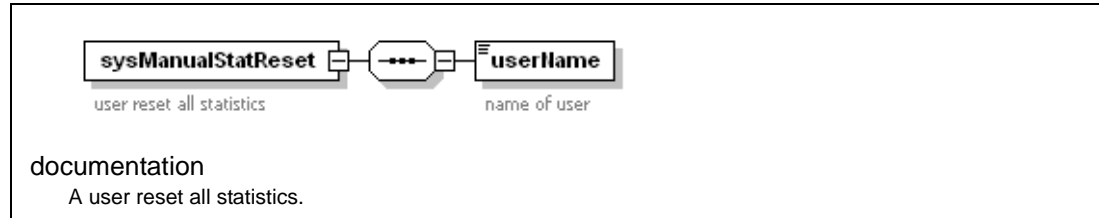


documentation
 User reset the manager.

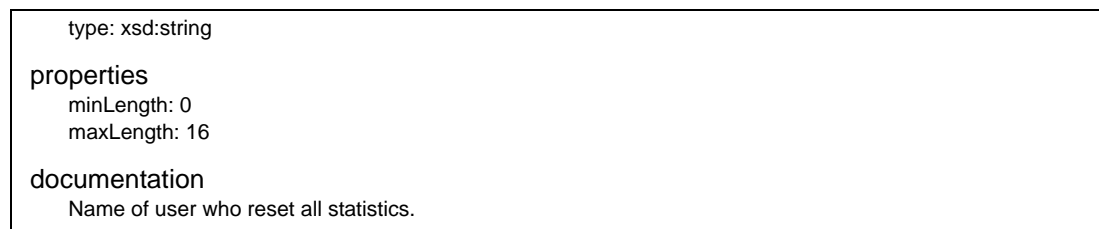
element `notifications/event/sysManualDccReset/username`

type: xsd:string
 properties
 minLength: 0
 maxLength: 16
 documentation
 Name of user who reset the manager.

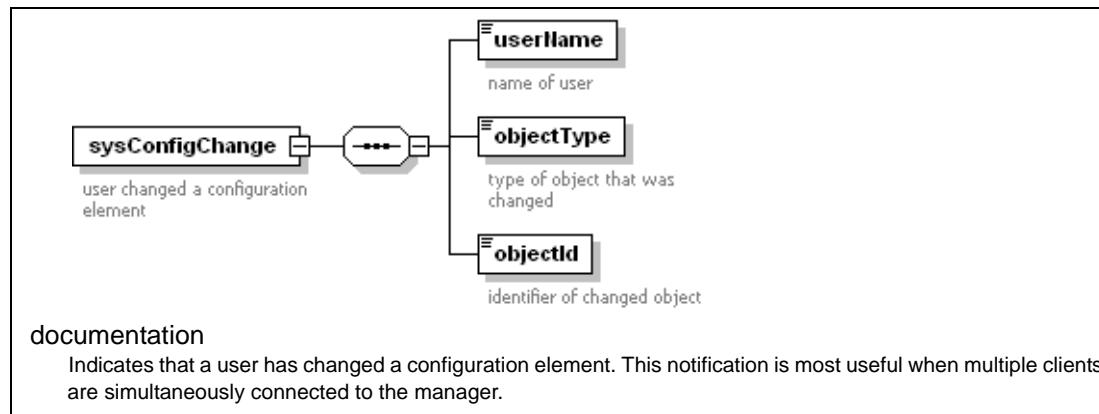
element `notifications/event/sysManualStatReset`



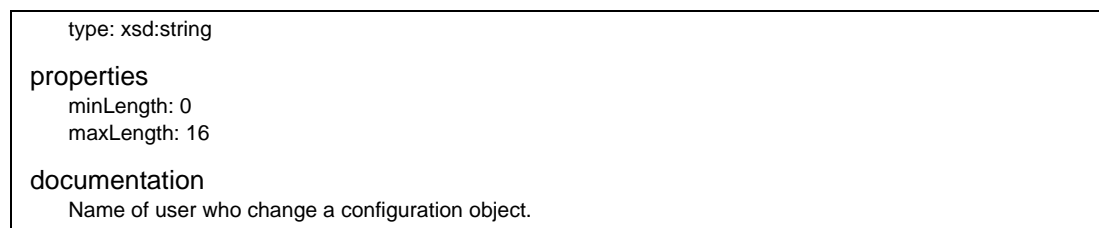
element `notifications/event/sysManualStatReset/username`



element `notifications/event/sysConfigChange`



element `notifications/event/sysConfigChange/username`



element `notifications/event/sysConfigChange/objectType`

<p>type: xsd:string</p> <p>properties</p> <p>enumerations:</p> <ul style="list-style-type: none"> mote path network networkId networkKey joinKey sessionKey sla system user channelBlackList security acl <p>documentation</p> <p>Type of configuration object that was changed.</p>

element `notifications/event/sysConfigChange/objectId`

<p>type: xsd:string</p> <p>documentation</p> <p>Identifier of object that was changed.</p>
--

element `notifications/event/sysCmdFinish`

<p>documentation</p> <p>A command finished executing.</p>

element notifications/event/sysCmdFinish/callbackId

type: xsd:unsignedLong
properties minInclusive: 0 maxInclusive:4294967295
documentation Callback Id associated with command.

element notifications/event/sysCmdFinish/objectType

type: xsd:string
properties enumerations: mote path network networkId networkKey joinKey sessionKey sla system user channelBlackList
documentation Type of command.

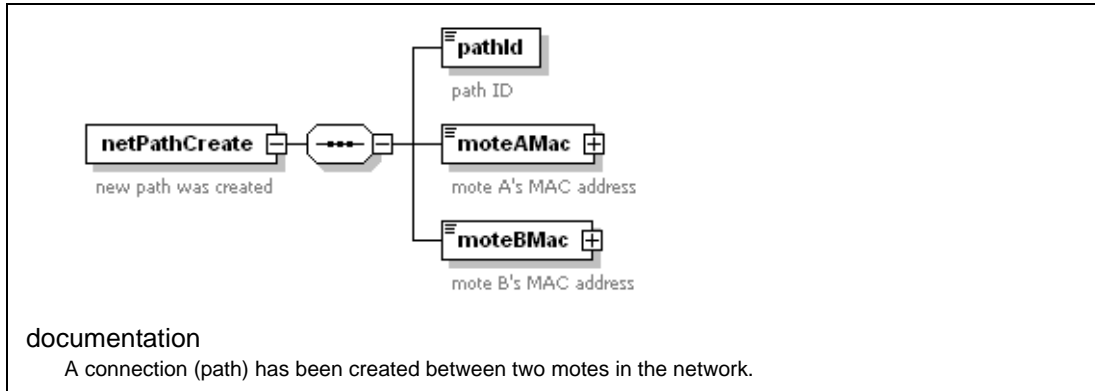
element notifications/event/sysCmdFinish/macAddress

type: xsd:string
documentation MAC address of mote.

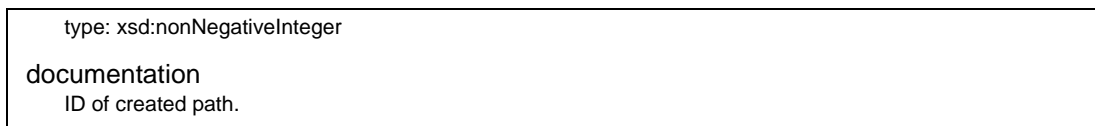
element notifications/event/sysCmdFinish/resultCode

type: xsd:integer
documentation Code indicating the command result.

element `notifications/event/netPathCreate`



element `notifications/event/netPathCreate/pathId`



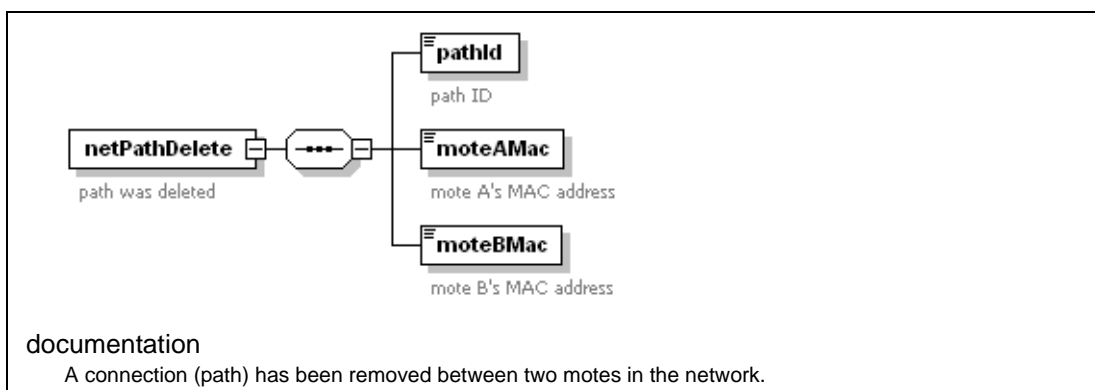
element `notifications/event/netPathCreate/moteAMac`



element `notifications/event/netPathCreate/moteBMac`



element `notifications/event/netPathDelete`



element `notifications/event/netPathDelete/pathId`

type: `xsd:nonNegativeInteger`

documentation
ID of deleted path.

element `notifications/event/netPathDelete/moteAMac`

type: `xsd:string`

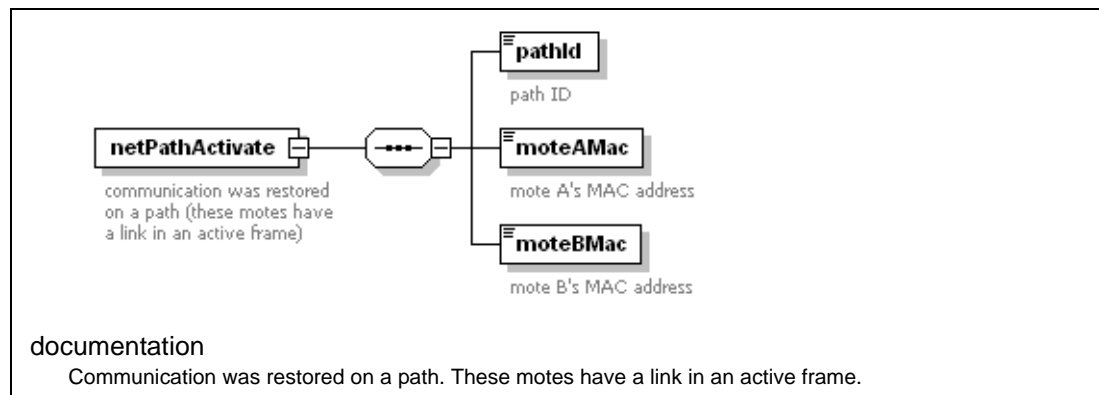
documentation
MAC address of mote on one end of path.

element `notifications/event/netPathDelete/moteBMac`

type: `xsd:string`

documentation
MAC address of mote on other end of path.

element `notifications/event/netPathActivate`



element `notifications/event/netPathActivate/pathId`

type: `xsd:nonNegativeInteger`

documentation
ID of path for which communication was restored.

element `notifications/event/netPathActivate/moteAMac`

type: `xsd:string`

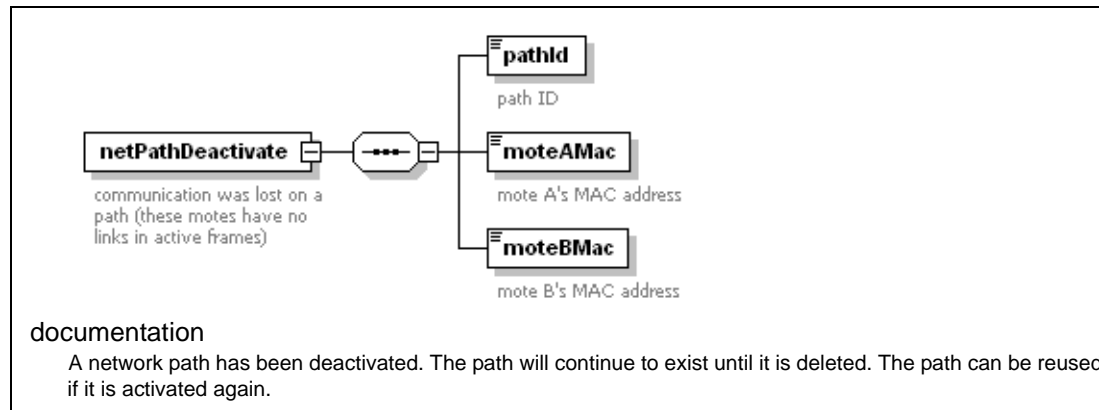
documentation
MAC address of mote on one end of the path for which communication was restored.

element notifications/event/netPathActivate/moteBMac

type: xsd:string

documentation

MAC address of mote on other end of the path or which communication was restored.

element notifications/event/netPathDeactivate**element notifications/event/netPathDeactivate/pathId**

type: xsd:nonNegativeInteger

documentation

ID of path that was deactivated.

element notifications/event/netPathDeactivate/moteAMac

type: xsd:string

documentation

MAC address of mote on one end of the path that was deactivated.

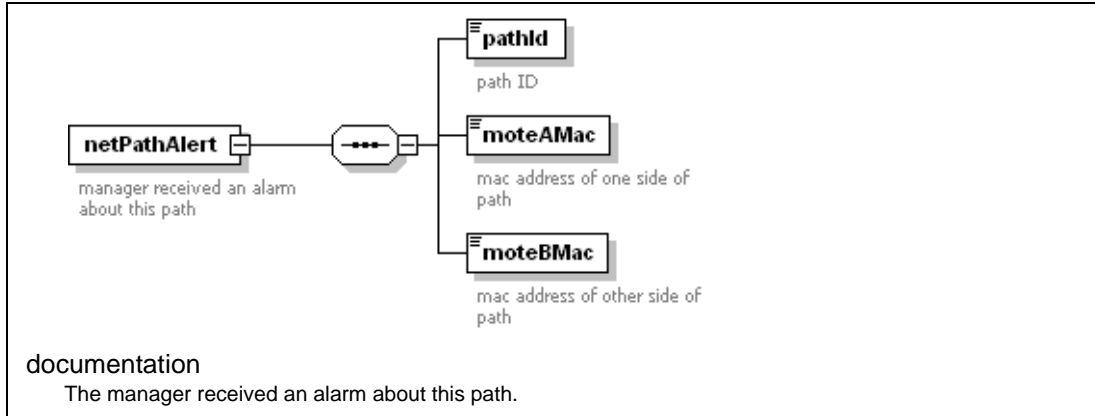
element notifications/event/netPathDeactivate/moteBMac

type: xsd:string

documentation

MAC address of mote on other end of the path that was deactivated.

element `notifications/event/netPathAlert`



element `notifications/event/netPathAlert/pathId`

type: `xsd:nonNegativeInteger`

documentation

ID of path for which the manager received an alarm.

element `notifications/event/netPathAlert/moteAMac`

type: `xsd:string`

documentation

MAC address of mote on one end of the path for which the manager received an alarm.

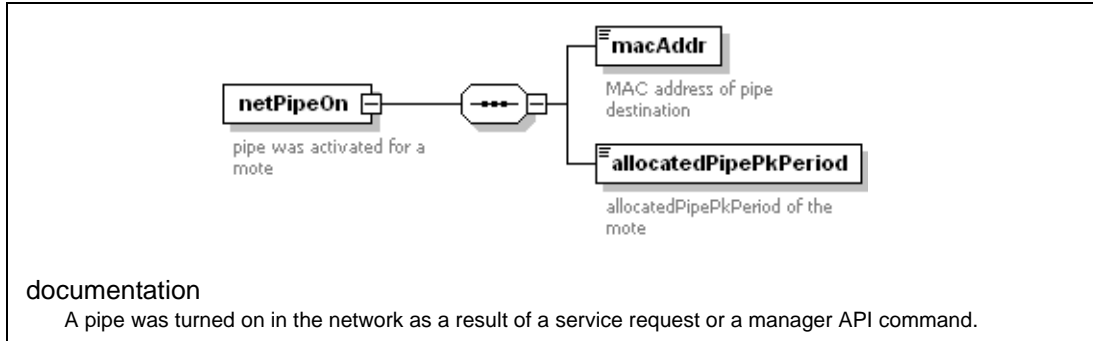
element `notifications/event/netPathAlert/moteBMac`

type: `xsd:string`

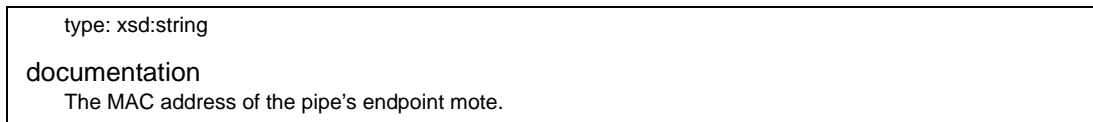
documentation

MAC address of mote on other end of the path for which the manager received an alarm.

element `notifications/event/netPipeOn`



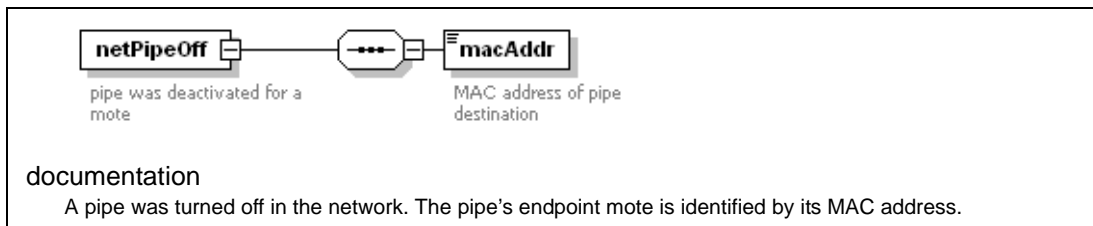
element `notifications/event/netPipeOn/macAddr`



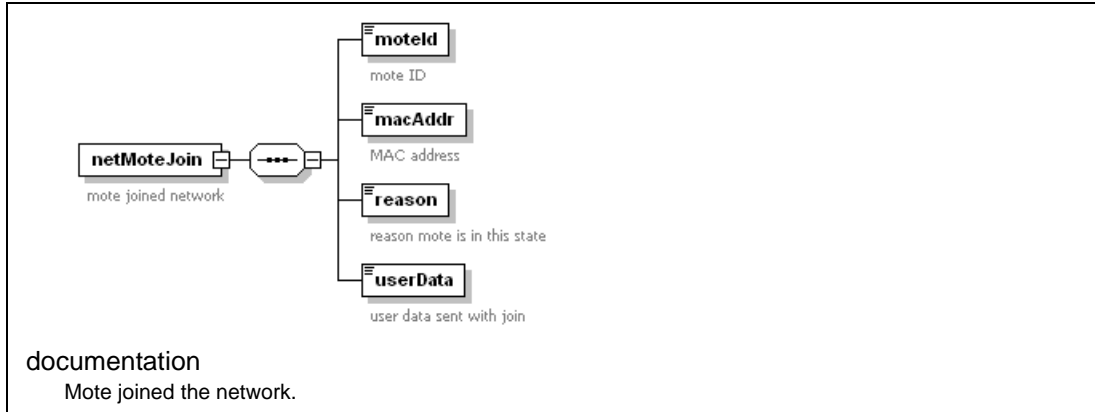
element `notifications/event/netPipeOn/allocatedPipePkPeriod`



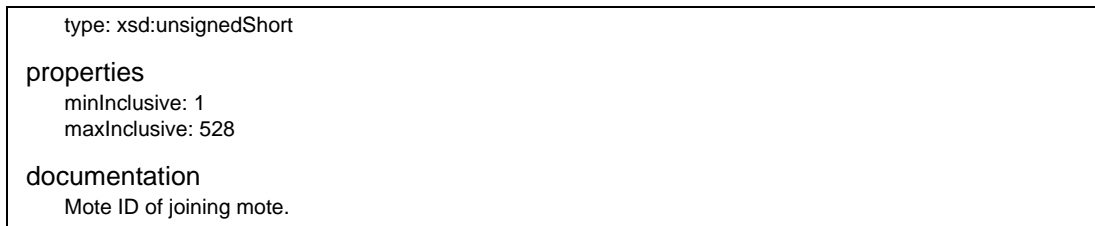
element `notifications/event/netPipeOff/macAddr`



element `notifications/event/netMoteJoin`



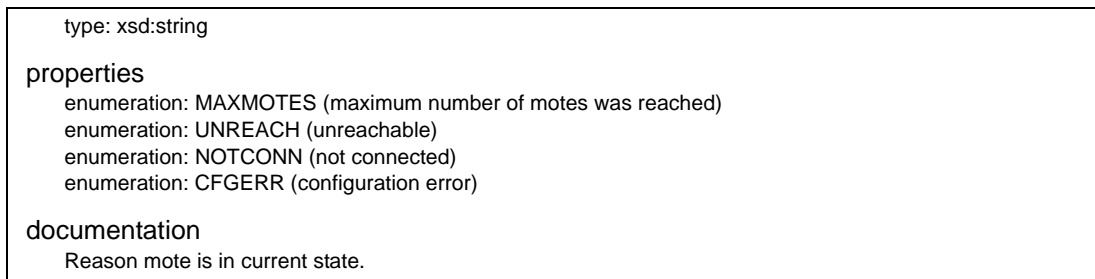
element `notifications/event/netMoteJoin/moteld`



element `notifications/event/netMoteJoin/macAddr`



element `notifications/event/netMoteJoin/reason`



element `notifications/event/netMoteJoin/userData`



element `notifications/event/netMoteLive`



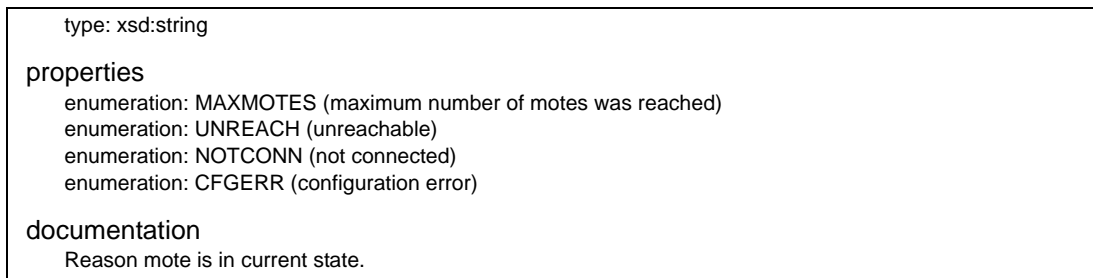
element `notifications/event/netMoteLive/motelId`



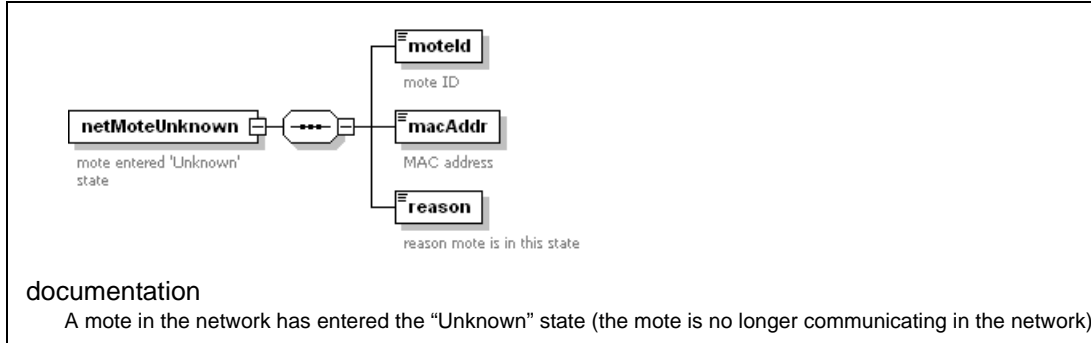
element `notifications/event/netMoteLive/macAddr`



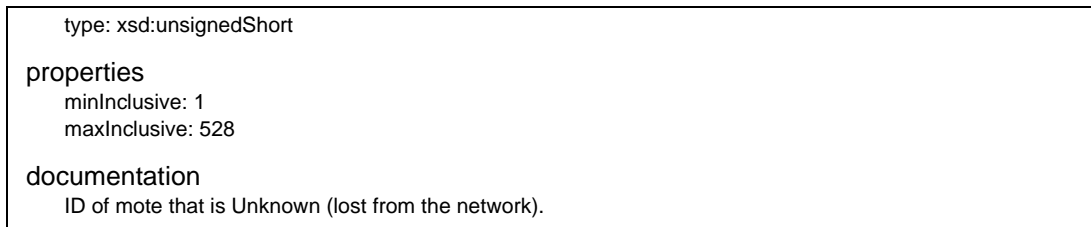
element `notifications/event/netMoteLive/reason`



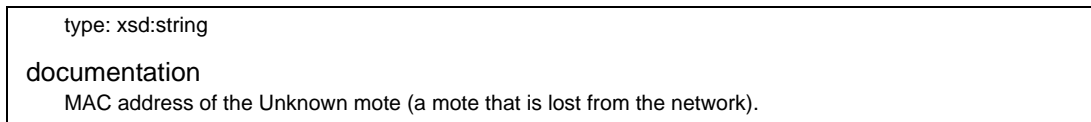
element `notifications/event/netMoteUnknown`



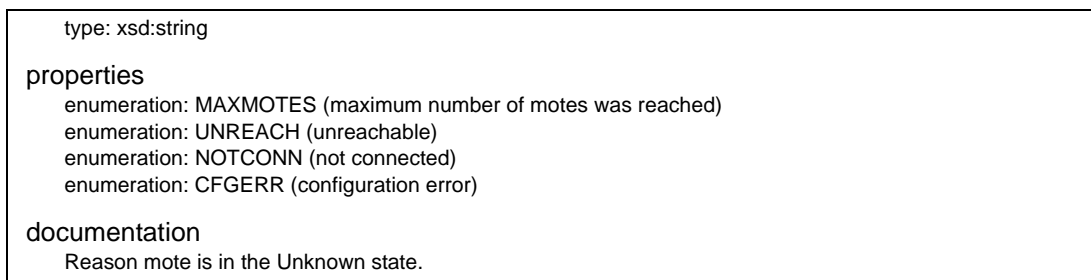
element `notifications/event/netMoteUnknown/moteld`



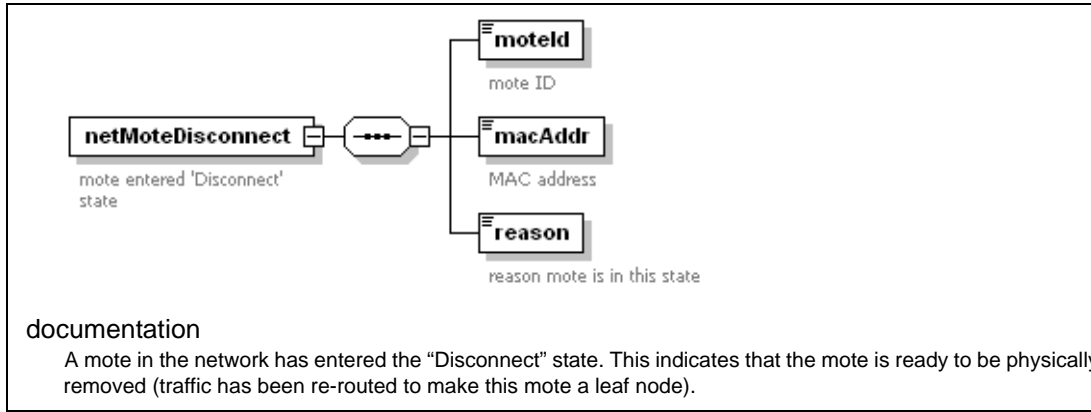
element `notifications/event/netMoteUnknown/macAddr`



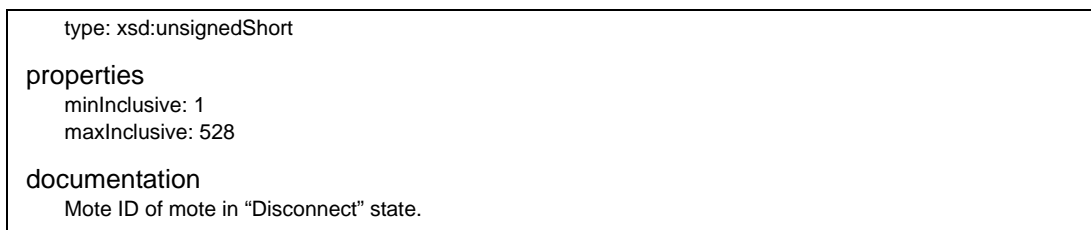
element `notifications/event/netMoteUnknown/reason`



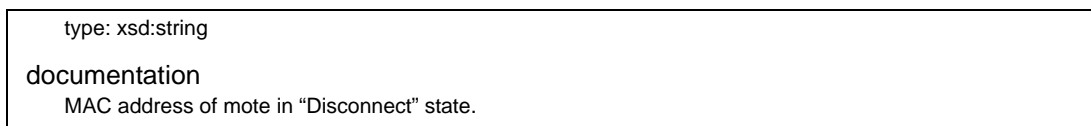
element `notifications/event/netMoteDisconnect`



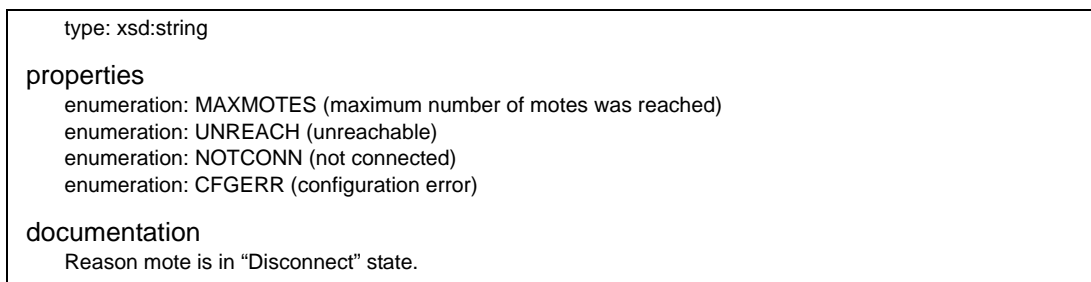
element `notifications/event/netMoteDisconnect/moteld`



element `notifications/event/netMoteDisconnect/macAddr`



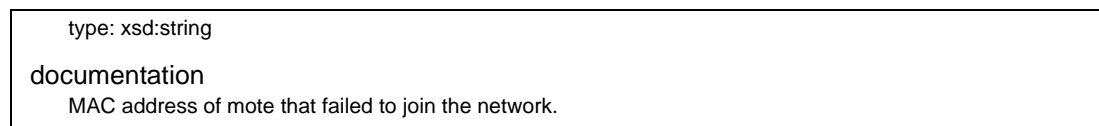
element `notifications/event/netMoteDisconnect/reason`



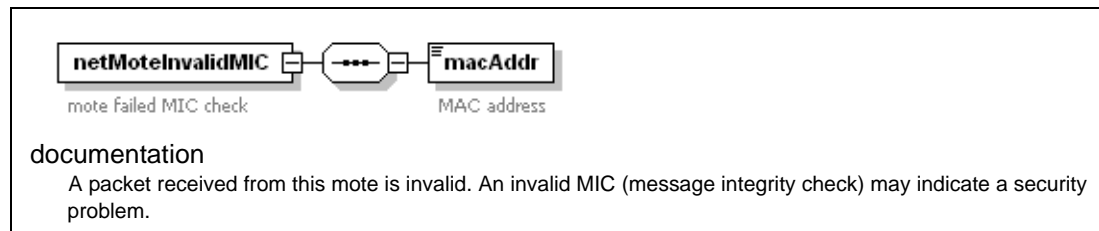
element `notifications/event/netMoteJoinFailure`



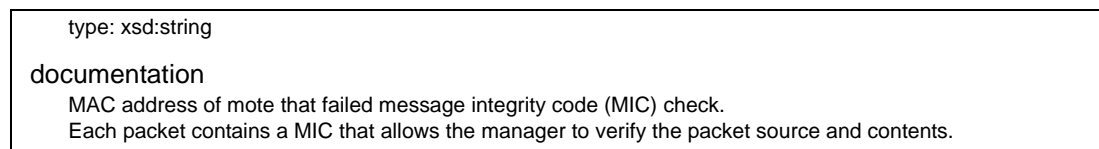
element `notifications/event/netMoteJoinFailure/macAddr`



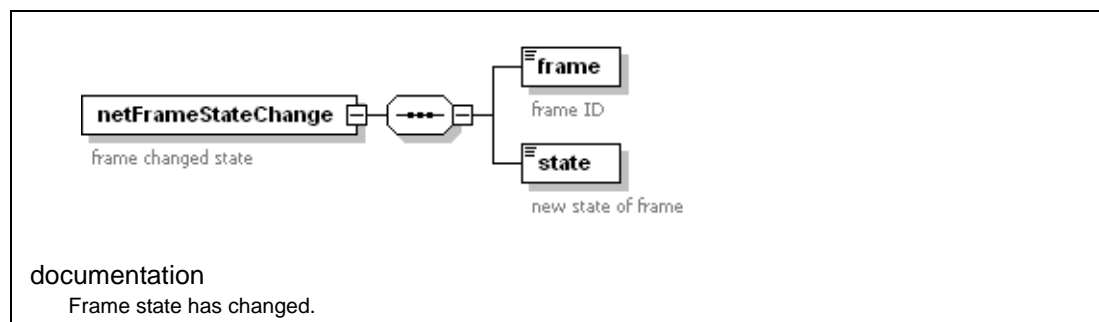
element `notifications/event/netMoteInvalidMIC`



element `notifications/event/netMoteInvalidMIC/macAddr`



element `notifications/event/netFrameStateChange`



element `notifications/event/netFrameStateChange/frame`

type: `xsd:nonNegativeInteger`

documentation

ID of the frame whose state has changed.

element `notifications/event/netFrameStateChange/state`

type: `xsd:string`

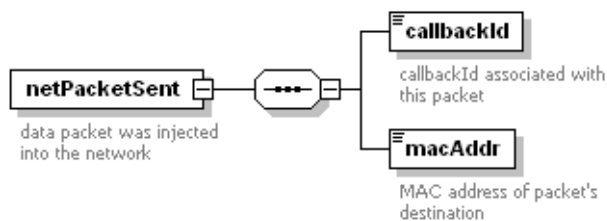
properties

enumeration: `on`
 enumeration: `off`
 enumeration: `activating`
 enumeration: `deactivating`

documentation

New state of frame.

element `notifications/event/netPacketSent`



documentation

When a `sendResponse` or `send Request` command is issued, a data packet is sent into the network and this notification is generated.

example

```
<event>
  <eventId>1234</eventId>
  <time>9999</time>
  <netPacketSent>
    <callbackId>9876</callbackId>
    <macAddr>00-00-00-00-00-00-00-01</macAddr>
  </netPacketSent>
</event>
```

element `notifications/event/netPacketSent/callbackId`

type: `xsd:nonNegativeInteger`

documentation

The `callbackId` associated with the data packet that was injected into the network.

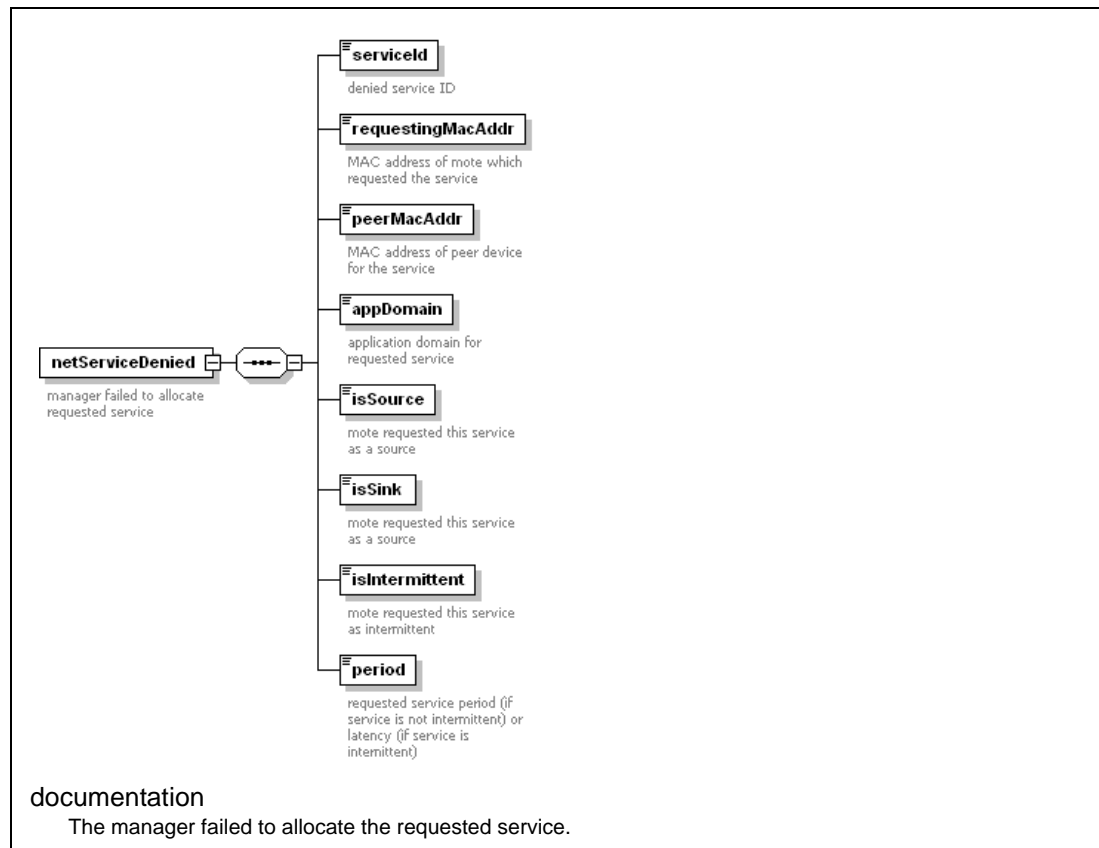
element `notifications/event/netPacketSent/macAddr`

type: `xsd:string`

documentation

The destination MAC address of the data packet that was injected into the network.

element `notifications/event/netServiceDenied`



element `notifications/event/netServiceDenied/serviceld`

type: `xsd:nonNegativeInteger`

documentation

Service ID

element `notifications/event/netServiceDenied/requestMacAddr`

type: `xsd:string`

documentation

MAC address of the mote that requested the service.

element notifications/event/netServiceDenied/peerMacAddr

<p>type: xsd:string</p> <p>documentation MAC address of peer device for the service.</p>
--

element notifications/event/netServiceDenied/appDomain

<p>type: xsd:string</p> <p>properties enumeration: publish enumeration: event enumeration: maintenance enumeration: blockTransfer (not used)</p> <p>documentation Application domain for the requested the service.</p>

element notifications/event/netServiceDenied/isSource

<p>type: boolean</p> <p>documentation True if the mote requested this service as a source.</p>
--

element notifications/event/netServiceDenied/isSink

<p>type: boolean</p> <p>documentation True if the mote requested this service as a source.</p>
--

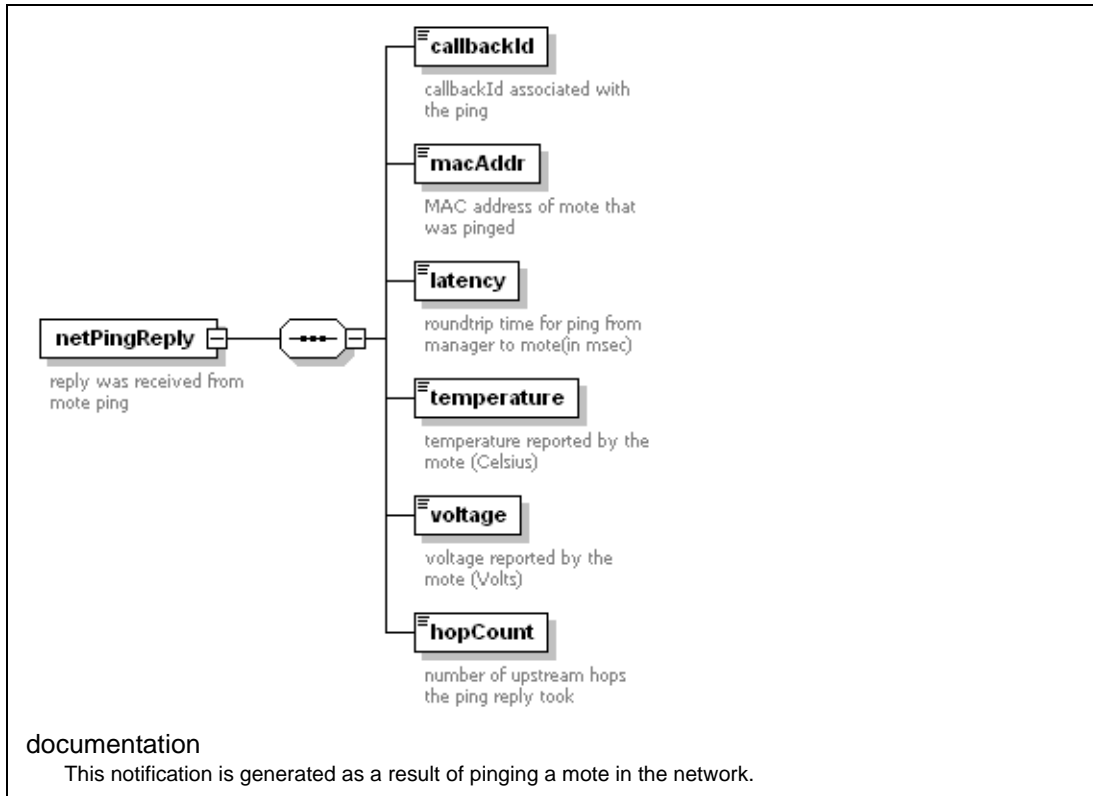
element notifications/event/netServiceDenied/isIntermittent

<p>type: boolean</p> <p>documentation True if the mote requested this service as intermittent.</p>
--

element notifications/event/netServiceDenied/period

<p>type: xsd:nonNegativeInteger</p> <p>documentation Requested service period (if service is not intermittent) or latency (if service is intermittent).</p>

element **notifications/event/netPingReply**



element **notifications/event/netPingReply/callbackId**

type: unsignedLong

properties
minInclusive: 0
maxInclusive: 4294967295

documentation
Callback Id associated with the ping.

element **notifications/event/netPingReply/macAddr**

type: xsd:string

documentation
MAC address of the mote that was pinged.

element notifications/event/netPingReply/latency

<p>type: unsignedLong</p> <p>properties</p> <p>minInclusive: 0 maxInclusive: 4294967295</p> <p>documentation</p> <p>Round-trip time (in msec) for ping from the manager to mote.</p>
--

element notifications/event/netPingReply/temperature

<p>type: xsd:float</p> <p>documentation</p> <p>Temperature (°C) reported by the mote.</p>
--

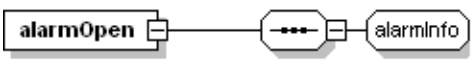
element notifications/event/netPingReply/voltage

<p>type: xsd:float</p> <p>documentation</p> <p>Voltage (volts) reported by the mote.</p>

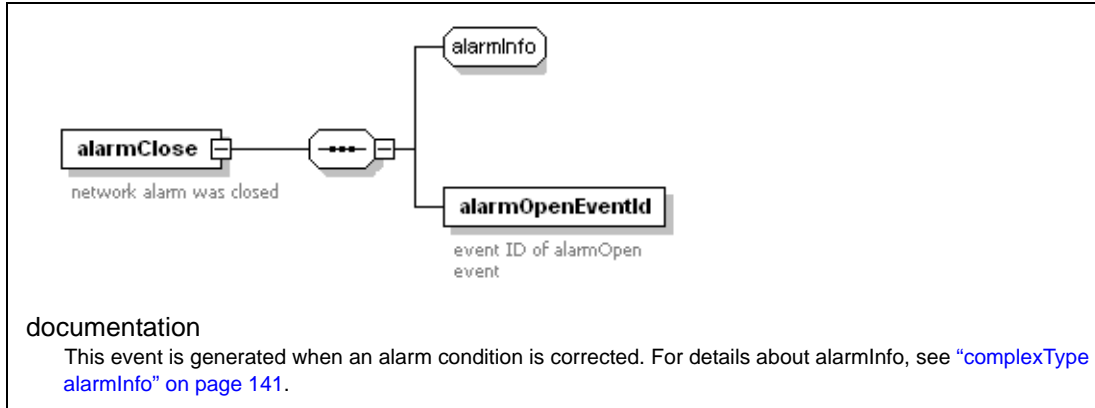
element notifications/event/netPingReply/hopCount

<p>type: xsd:short</p> <p>properties</p> <p>minInclusive: -128 maxInclusive: 127</p> <p>documentation</p> <p>Number of upstream hops the ping reply took.</p>

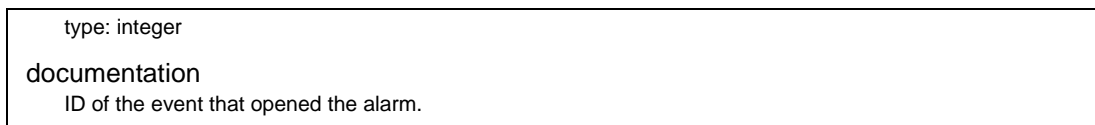
element notifications/event/alarmOpen

 <p>The diagram shows a rectangular box labeled 'alarmOpen' with a small square on its right side. A line connects this box to a hexagonal box labeled 'alarmInfo', also with a small square on its right side. This represents an element containing another element.</p> <p>network: alarm was opened</p>
<p>documentation</p> <p>Network alarm was opened. For schema details see "complexType alarmInfo" on page 141.</p>

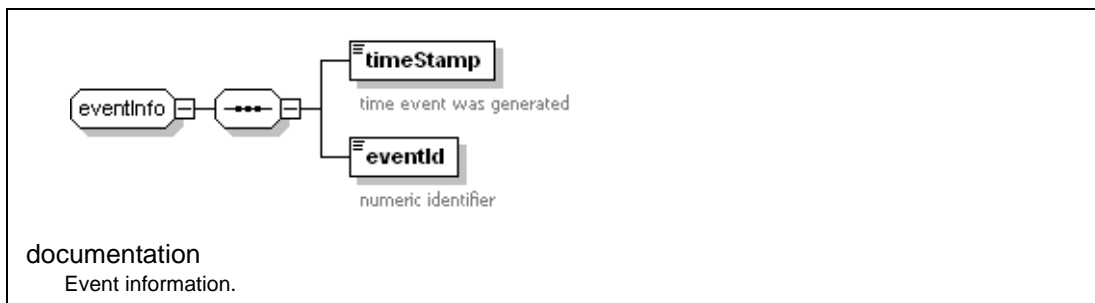
element `notifications/event/alarmClose`



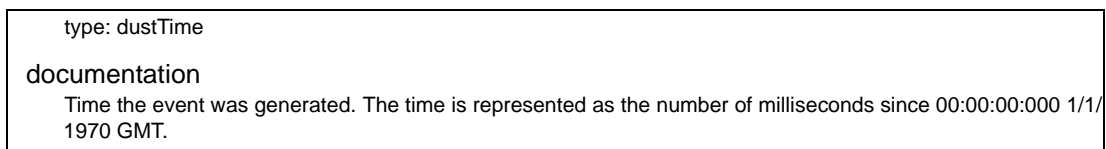
element `notifications/event/alarmClose/alarmOpenEventId`



element `notifications/event/eventInfo`

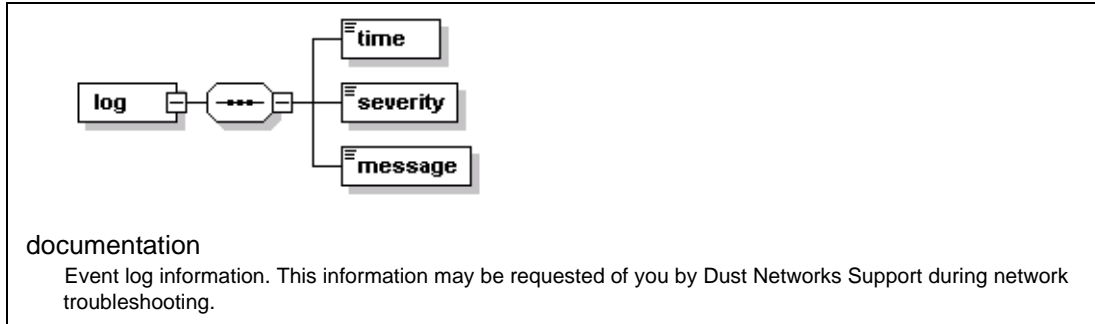
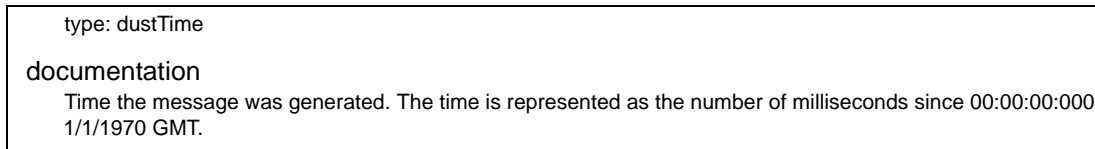


element `notifications/event/eventInfo/timeStamp`

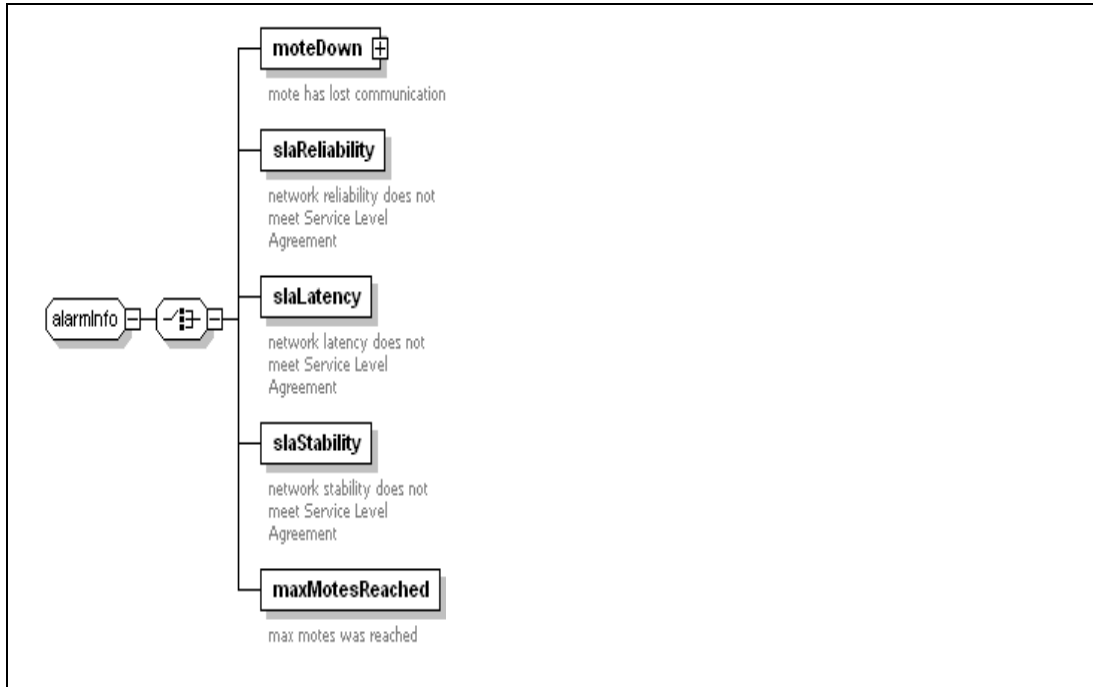


element `notifications/event/eventInfo/eventId`



element notifications/log**element notifications/log/time****element notifications/log/severity****element notifications/log/message**

complexType **alarmInfo**



element **alarmInfo/moteDown/macAddr**

type: xsd:string

documentation
 This alarm is generated when a mote (identified by its MAC address) fails to respond to message from the manager.

element **alarmInfo/slaReliability**

documentation
 This alarm is generated when network reliability drops below the preset SLA threshold (see ["element config/ Network/Slas" on page 72](#)).

element alarmInfo/slaLatency**documentation**

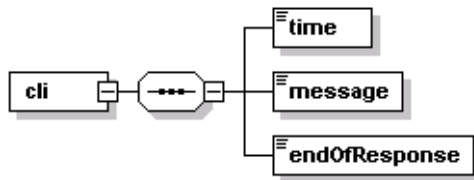
This alarm is generated when network latency goes above the preset SLA threshold (see ["element config/Network/Sla" on page 72](#)).

element alarmInfo/slaStability**documentation**

This alarm is generated when network stability drops below the preset SLA threshold (see ["element config/Network/Sla" on page 72](#)).

element alarmInfo/maxMotesReached**documentation**

This alarm is generated when the network reaches the maximum number of motes for which the network has been configured (see ["element config/Network/maxMotes" on page 67](#)).

element notifications/cli**documentation**

This event is sent in response to the CLI command (see ["cli" on page 35](#)). Each CLI notification contains a line of the CLI output.

example

```

<cli>
  <time>1210978990856</time>
  <message><![CDATA[minNetReliability:      99.00]]></message>
</cli>
<cli>
  <time>1210978990863</time>
  <message><![CDATA[maxNetLatency:        12500]]></message>
</cli>
<cli>
  <time>1210978990870</time>
  <message><![CDATA[minNetPathStability:   50.00]]></message>
</cli>
<cli>
  <time>1210978990894</time>
  <message><![CDATA[
]]></message>
  <endOfResponse/>
  
```

element notifications/cli/time

type: xsd:positiveInteger

documentation

Time output was generated.

element notifications/cli/message

type: xsd:string

documentation

One line of cli output.

Output is sent line by line. Each line has a maximum of 80 characters. Lines longer than 80 characters are split into multiple notifications.

Index

A

- activateAdvertising
 - command reference, 33
- activateFastPipe, 6, 32
- alarm
 - network latency, 142
 - network reliability, 141
 - network stability, 142
- alarmInfo, 141
 - moteDown, 141
 - slaLatency, 142
 - slaReliability, 141
 - slaStability, 142
- Alarms configuration element, 108
 - eventId, 109
 - timeStamp, 109
- authorization, 3
 - notification channel, 24
 - notification channel requests, 11

B

- battery voltage, 94, 99

C

- cancelOtap, 6, 32
 - command reference, 35
- cli, 32
 - command reference, 35
- command-line interface session, 65
- commands, 2, 5, 6, 10, 23
 - reference, 31
 - sending to the Manager, 22
- concurrent client support, 3
- configuration schema
 - description, 5
- control channel, 2, 7, 10, 32
 - logging in, 12–13
 - logging out, 13
 - overview, 2
 - using, 12
- createConfig, 6, 32
 - adding an object, 19
 - command reference, 36

- current time, 65

D

- data packets
 - maximum allowable latency, 72
 - minimum allowable reliability, 72
 - timestamp between transmission and receipt, 99
- deactivateFastPipe, 6, 32
- decommissionDevice, 6, 32
 - command reference, 37
- deleteConfig, 6, 32
 - command reference, 38
- depth, 14
- document string, 31

E

- Event Log configuration element, 110
- exchange join key
 - command reference, 37, 43, 44
- exchangeJoinKey, 6, 32
- exchangeJoinkey
 - command reference, 39
- exchangeMoteJoinKey, 6, 32
- exchangeMoteJoinkey
 - command reference, 40
- exchangeNetworkId, 6, 32
 - command reference, 42
- exchangeNetworkKey
 - command reference, 43
- exchangeSessionKey, 6, 32
 - command reference, 44

G

- getConfig, 6, 32
 - command reference, 45
 - specifying depth, 14–17
- getLatency, 6, 32
 - command reference, 46
- getTime, 7, 32
 - command reference, 47

H

hardware model, 92
 HTTP wrapper, 10, 12

L

login, 6, 7, 32
 command reference, 47
 login token, 31
 logout, 7, 13, 32
 command reference, 48

M

MAC address, 90, 113
 joining mote, 129, 130, 131, 132
 mote
 firmware version, 93
 hardware model, 92
 hardware version, 93
 state, 94
 mote ID, 113
 Motes configuration elements
 advertisingStatus, 96
 allocatedPkPeriod, 95
 dischargeCurrent, 91
 dischargeTime, 91
 enableRouting, 92
 hwModel, 92
 hwRev, 93
 isAccessPoint, 93
 joinTime, 94
 macAddr, 90
 moteId, 90
 name, 91
 needNeighbor, 96
 numJoins, 93
 numNeighbors, 95
 pipeStatus, 95
 powerSource, 91
 productName, 92
 reason, 94
 recoveryTime, 92
 requestedPipePkPeriod, 95
 state, 94
 Statistics, 97
 avgLatency, 99
 chargeConsumption, 99
 lifetime, 102
 numJoins, 99
 reliability, 98, 99
 stat15MinSet, 99
 stat1DaySet, 101
 statCur, 98
 temperature, 99
 swRev, 93
 voltage, 94

N

network
 encryption password for joining, 87
 ID, 67
 latency, 82
 name, example of changing, 20
 reliability, 81
 network configuration elements
 accessPointPA, 68
 bandwidthProfile, 69
 ccaEnabled, 68
 ChannelBlackList, 73
 maintEndTime, 71
 maintStartTime, 70
 manualAdvFrameSize, 70
 manualDSFrameSize, 70
 manualUSFrameSize, 70
 maxMotes, 67
 minPipePkPeriod, 69
 minServicesPkPeriod, 69
 netName, 67
 netQueueSize, 71
 networkId, 67
 numMotes, 68
 optimizationEnable, 67
 OtapStatus, 74
 requestedBasePkPeriod, 68
 Sla, 72
 Statistics, 80
 lifetime, 85
 lostUpstreamPackets, 82
 netLatency, 82
 netPathStability, 82
 netReliability, 81
 NetStatGroup, 81
 stat15MinSet, 82
 stat1DaySet, 83
 statCur, 81
 userQueueSize, 71
 notification channel
 authorization, 24
 logging out, 48
 message format, 11
 overview, 3
 schema reference, 111
 subscribing, 23–24, 55, 56
 notification schema
 description, 6
 reference, 111
 notifications configuration elements, 111
 cli, 142
 data, 112
 dataInfo, 112
 payload, 112
 event, 114
 alarmClose, 139
 alarmOpen, 138
 channel, 116
 eventInfo, 139
 netFrameStateChange, 133

- netMoteDisconnect, 132
 - netMoteInvalidMIC, 133
 - netMoteJoin, 129
 - netMoteJoinFailure, 133
 - netMoteLive, 130
 - netMoteUnknown, 131
 - netPacketSent, 134
 - netPathActivate, 125
 - netPathAlert, 127
 - netPathCreate, 124
 - netPathDeactivate, 126
 - netPathDelete, 124
 - netPingReply, 137
 - netPipeOff, 128
 - netPipeOn, 128
 - netReset, 115
 - netServiceDenied, 135
 - sysBootUp, 115
 - sysCmdFinish, 122
 - sysConfigChange, 121
 - sysConnect, 115
 - sysDisconnect, 116
 - sysManualDccReset, 120
 - sysManualMoteDecommission, 119
 - sysManualMoteDelete, 118
 - sysManualMoteReset, 117
 - sysManualNetReset, 120
 - sysManualStatReset, 121
 - userName, 116
 - log, 140
- O**
- OtapStatus configuration elements
 - curFileCurrentRetry, 76
 - curFileSentFragments, 77
 - curFileTotalRetries, 76
 - currentFile, 76
 - Files, 77
 - Motes, 77, 78, 79
 - state, 75
 - timeToCommit, 75
 - uploadedDevices, 75
- P**
- path
 - minimum allowable stability, 72
 - stability, 82
 - Path configuration elements, 103
 - moteAMac, 103
 - moteBMac, 104
 - numLinks, 104
 - pathId, 103
 - Statistics, 104
 - abPwr, 105
 - baPwr, 105
 - lifetime, 108
 - PathStatGroup, 105
 - stability, 106
 - stat1Day, 107
 - statCur, 105
 - pingMote, 7, 32
 - power source, 91
- R**
- received signal strength indication, 105
 - reset, 7, 32
 - example of reset command, 23
 - reset object
 - command reference, 49
- S**
- security configuration elements, 86
 - acceptHARTDevicesOnly, 87
 - ACL, 87
 - commonJoinKey, 87
 - Device joinKey, 88
 - Device macAddress, 88
 - securityMode, 87
 - security keys, 22
 - sending binary data, 50
 - sendRequest
 - command reference, 50
 - sendResponse
 - command reference, 52
 - setConfig, 7, 32
 - changing an object, 20
 - command reference, 53
 - Sla configuration elements
 - maxNetLatency, 72
 - minNetPathStability, 72
 - minNetReliability, 72
 - SmartMesh API
 - commands overview, 6
 - scripting, 9
 - SmartMesh manager API description, 5
 - SmartMesh manager description, 2
 - startOtap, 7, 32
 - startOtap
 - command reference, 54
 - subscribe, 7, 32
 - command reference, 55
 - system configuration objects
 - cliTimeout, 65
 - controllerSwRev, 65
 - ctrlSslPort, 62
 - ctrlTcpPort, 62
 - dataSslPort, 63
 - dataTcpPort, 63
 - discoveryUpdPort, 63
 - hwModel, 64
 - hwRev, 64
 - location, 62
 - serialNumber, 64
 - sslEnabled, 63
 - startTime, 65
 - swRev, 64
 - systemName, 62
 - time, 65

T

TCP port
 control channel, 62
 notification channel, 63
TCP/IP, 12, 24
timestamp
 mote join time, 94

U

unsubscribe
 command reference, 56
user
 password, 86
 privilege level, 86
user configuration elements
 password, 86
 privilege, 86
 User, 85
 userName, 85

X

XML, 3
 overview, 3
 syntax, 11
XML-RPC, 4, 12
 overview, 4