OpenSM User's Manual

Rev 1.11

OpenSM User's Manual

Document Number: 2277UM

Mellanox Technologies, Inc.
2900 Stender Way
Santa Clara, CA 95054
U.S.A.
www.Mellanox.com

Tel: (408) 970-3400
Fax: (408) 970-3403

Mellanox Technologies Ltd
PO Box 586 Hermon Building
Yokneam 20692
Israel

Tel: +972-4-909-7200
Fax: +972-4-959-3245

Table of Contents

About this Manual

This manual describes the OpenSM package. OpenSM is a Subnet Manager for the initialization of InfiniBand compliant devices.

This manual is organized in the following manner:

- Chapter 1 provides an overview of the OpenSM package(page 7)
- Chapter 2 provides instructions for the installation of the OpenSM package (page 9)
- Chapter 3 describes how to use the tools included this OpenSM package (page 11)
- Chapter 4 describes the UPDN algorithm, its purpose and usage (page 31)

Intended Audience

The target audience of this User's Manual is System Administrators who have installed InfiniBand hardware and need to run a Subnet Manager (SM) in order to initialize it.

Related Documentation

For InfiniBand related issues, please refer to the following specification:

- InifiniBand Architecture Specification Volume 1, Release 1.2

For implementing user-level applications that interface with the InfiniBand Subnet Administrator, send MADs through the SMI or GSI, please refer to the following manual:

- OpenSM Vendor Layer API and Programmer's Manual, Rev. 0.1

Conventions

Throughout this document, TCL shell commands are typed in the following format:

```
command
=> ...
```

where the second line is the command return.

For Example:

```
osm_opts configure
=> { -m_key -sm_key -subnet_prefix -m_key_lease_period -sweep_interval -max_wire_smps
-transaction_timeout -sm_priority -lmc -max_op_vls -reassign_lids -reassign_lfts -ignore_other_sm
-single_thread -no_multicast_option -disable_multicast -force_log_flush -subnet_timeout
-packet_life_time -head_of_queue_lifetime -local_phy_errors_threshold -overrun_errors_threshold
-polling_timeout -polling_retry_number -force_heavy_sweep -sweep_on_trap -max_port_profile
-port_profile_switch_nodes -updn_activate -updn_guid_file }
```

1 Overview

The *InifiniBand Architecture Specification (Volume 1)* defines extensively and explicitly how InfiniBand (IB) compliant devices are to be managed. There it is stated that routing and other management policies, for each compliant device, should be exported to a centralized entity called: Subnet Manager (SM). This standardization of management significantly reduces the costs of hardware when compared to the costs of traditional distributed network management policies.

OpenSM is an InfiniBand compliant Subnet Manager. It is provided in two flavors: a fixed flow executable called ***opensm***, and a fully configureable version called ***osmsh***. The latter is provided as a TCL extension package called ***osm***. Both ***opensm*** and ***osm*** are accompanied by a testing application called ***osmtest***.

The target audience of this User's Manual is System Administrators who have installed InfiniBand hardware and need to run a Subnet Manager in order to initialize it. Since OpenSM implements an SM as defined in the IB specification, it will not be defined here again. Instead, to understand what OpenSM implements, the user is kindly referred to the following chapters of that specification: Management Model (13), Subnet Management (14), and Subnet Administration (15).

This User's Manual describes the OpenSM package in the following sections:

1.1 Key Concepts and Terms

Throughout this manual, there is frequent reference to various concepts and terms which are common to the general audience of System Administrators. There is also reference to InfiniBand-specific concepts and terms, a part of which are briefly defined in the list to follow. Other terms may be defined where relevant in this document; however, the IB specification remains the main reference for all (missing and existing) IB terms.

- **IB devices:**
  Integrated Circuits implementing InfiniBand compliant communication.
- **IB Fabric/Cluster/Subnet:**
  A set of IB devices connected by IB cables.
- **Subnet Manager (SM):**
  One of several entities involved in the configuration and control of the subnet.
- **Master Subnet Manager:**
  The subnet manager that is authoritative, that has the reference configuration information for the subnet.
- **Standby Subnet Manager:**
  A subnet manager that is currently quiescent, and not in the role of a master SM, by agency of the master SM.
- **Subnet Administrator (SA):**
  An application (normally part of the Subnet Manager) that implements the interface for querying and manipulating subnet management data.

- **LID**:
  An address assigned to a port (data sink or source point) by the Subnet Manager, unique within the subnet, used for directing packets within the subnet.
- **Unicast Linear Forwarding Tables (LFT):**
  A table that exists in every switch providing the port through which packets should be sent to each LID.
- **Multicast Forwarding Tables:**
  A table that exists in every switch providing the list of ports to forward received multicast packet. The table is organized by MLID.

1.2 Contents of OpenSM Package

The OpenSM package contains the following executables and libraries:

- ***opensm*:**
  A Subnet Manager and Administrator. It should be used for standard simple installations.
- ***osmsh*:**
  A Tcl extension provided as a package called ***osm***, which is extended with the full set of ***opensm*** commands and its data model. It should be used in case non-standard policies for Quality of Service, LID assignment, routing, etc., are needed.
- ***osmtest*:**
  A simple application to test ***opensm*** and ***osmsh***. It is capable of exercising most of the SA queries and provide clear feedback on their success.
- ***osmv_svc* shared library:**
  Contains services which enable user-level applications to interface with the SA, send MADs through the SMI or GSI. An extensive API definition is provided by the document *OpenSM Vendor Layer API and Programmer's Manual, Rev. 0.1*

1.3 Dependencies

OpenSM in its current form is available on top of VAPI and OpenIB stacks. ***osmsh*** (the ***osm*** package) is dependent upon TCL revision 8.3 or later.

2 Installation of OpenSM Package

OpenSM is bundled into the releases of VAPI, OpenIB (IB Gold Distribution) and various other distributions. Therefore, it is very possible that OpenSM is already installed on the user's machine. However, for new, separate releases of OpenSM, stand-alone installation instructions are provided below.

2.1 Installation Prerequisites

Before the installation of the OpenSM package, it is necessary to guarantee the following requirements:

- Either a VAPI or an OpenIB driver is available on the system.
- The latest OpenSM release has been downloaded from Mellanox's docs.mellanox.com or as part of the IB Gold Distribution package (via www.mellanox.com).
- In case you wish to have access to the ***osmsh***, Tcl8.3 or Tcl8.4 needs to be installed. Standard Linux installations normally include this package. However, if you are unable to run the command "tclsh8.3" or "tclsh8.4" on your machine, you will need to download a Tcl package from one of the sources making it available on the WWW.

2.2 Installing the OpenSM Software

As mentioned before, OpenSM may be installed on top of the OpenIB stack or on top of VAPI. Installation instructions for both options are provided below. The following routines assume you were able to obtain the OpenSM tar gzip archive named: osm-XXXX.tgz.

2.2.1 Installation on top of Standard Mellanox IB Stack Distributions

1. Make sure you know what type of standard installation exists on your system:
   - If it is the IB Gold Distribution, then it is installed by default under /usr/local/ibgd. Set the following two environment variables:
     - TSHOME: If /lib/modules/`uname -r`/source exists, then set TSHOME to /lib/modules/`uname -r`/source/drivers/infiniband/include. Else, set TSHOME to /lib/modules/`uname -r`/build/drivers/infiniband/include.
     - MTHOME: Set this variable to be /usr/local/ibgd/driver/infinihost/

     The IB Gold Distribution requires a build with the option: *-b ibgd*
   - If it is VAPI, then it is installed by default under /usr/mellanox (look for /usr/mellanox/include/vapi.h)

     Requires *build* option: *-b mtl*
2. Execute the following command: tar xvfz osm-XXX.tgz
3. Use the distribution type in the command: osm-XXX/install.sh -b ibgd *or* osm-XXX/install.sh -b mtl

2.2.2 Non-standard OpenIB Based Installation (Not Part of Mellanox Distribution)

You need to find where the VAPI and OpenIB C-header files (vapi.h and ts_ib_useraccess.h) are installed, and set the two environment variables TSHOME and MTHOME as described in the following instructions:

1. Define the environment variable TSHOME to point to the directory where the file ts_ib_useraccess.h resides.
2. Define the environment variable MTHOME to point to the directory above the 'VAPI include' subdirectory. Make sure that you can see the file $MTHOME/include/vapi.h.
3. Execute the following command: tar xvfz osm-XXX.tgz

4. Execute the following command: osm-XXX/install.sh --vendor=ibgd

2.2.3 Non-standard VAPI Based Installation (Not Part of Mellanox Distribution)

If you have installed VAPI in a non-standard manner (i.e. not under /usr/mellanox), perform the following steps:

1. Define the environment variable MTHOME to point to the directory above the VAPI include subdirectory, such that the file $MTHOME/include/vapi.h exists.
2. Execute the following command: tar xvfz osm-XXX.tgz
3. Execute the following command: osm-XXX/install.sh --vendor=mtl

3 Using *opensm*, *osmsh* and *osmtest*

This section of the manual describes the provided executables and their usage.

3.1 opensm

opensm is a simple command line executable that serves as both a Subnet Manager and a Subnet Administrator. It can be run with or without specifying any command line options.

3.1.1 Default or Common Case Usage

By entering ***opensm*** on the command line, without any additional options, the default settings will be chosen. These defaults were designed to meet the common case usage on clusters with up to a few hundred nodes. Thus, in this default mode, ***opensm*** will scan the IB fabric, initialize it, and sweep occasionally for changes.

OpenSM attaches to a specific IB port on the local machine and configures only the fabric connected to it. (If the local machine has other IB ports, OpenSM will ignore the fabrics connected to those other ports). In default operation mode, OpenSM will present the available ports and prompt for a port number to attach to. The run will be logged into two files: /var/log/messages and /tmp/osm.log. The first file will register only general major events, whereas the second will include details of reported errors. All errors reported in this second file should be treated as indicators of IB fabric health issues. (Note that when a fatal and non-recoverable error occurs, ***opensm*** will abort.)

Both log files should include the message "SUBNET UP" if ***opensm*** was able to setup the subnet correctly.

3.1.2 Non-Default Usage

It is possible for the user to run ***opensm*** with settings other than the default ones. Table 1 lists the ***opensm*** command line options in the first column, the effect of each option in the second, and tips on when to use each option in the last.

Table 1 - ***opensm*** Command Line Options.

Option	Effect	When to Use Option
-g <GUID in hex> --guid <GUID in hex>	OpenSM will bind to the port with the provided GUID. Default is to present to user the available GUIDs, and to select one of them.	To avoid the interaction required in order to select the port. It is possible to enter: "echo 1 | opensm" to select the first GUID.
-s <interval> --sweep <interval>	This option specifies the number of seconds between subnet sweeps. Specifying -s 0 disables sweeping. Default: OpenSM sweeps with intervals of 10 seconds.	To minimize unneeded sweeps, it is possible to set this value to 0. This will cause a single sweep and only traps will cause new sweeps. On large clusters, it is recommended to set this to a value higher than 60.
-t <milliseconds> --timeout <milliseconds>	This option specifies the time in milliseconds used for transaction timeouts (request to response). Default: Timeout value is 100ms.	This value should be changed only on large subnets. A reasonable value for a >1000nodes cluster is ~1000ms.
-p <PRIORITY> --priority <PRIORITY>	This option specifies the SM's PRIORITY. This will affect the handover cases, where master is chosen by priority and GUID.	Only if there is a need to explicitly control which SM should be the master.
-v --verbose	This option increases the log verbosity level. The -v option may be specified multiple times to further increase the verbosity level.	The first -v will print to the stdout a summary table of the discovered fabric.

Table 1 - ***opensm*** Command Line Options.

Option	Effect	When to Use Option
-V	This option sets the verbosity to its maximum level and forces log flushing.	Use this option to investigate an error or send a bug report
-f <file_name> --log_file <file_name>	This option defines the log file. By default the log goes to /tmp/osm.log. To send it to standard output use "-f stdout".	For your convenience only. Note that if you use -V the log file might be too large for the /tmp...
-o --once	This option causes OpenSM to configure the subnet once, then exit. Ports remain in the ACTIVE state.	For testing purposes only.
-r --reassign_lids	This option causes OpenSM to reassign LIDs to all endnodes. Specifying -r on a running subnet may disrupt subnet traffic. Default: OpenSM attempts to preserve existing LID assignments resolving multiple use of same LID.	Should rarely be used. Use only if re-numbering of all the LIDs is required. Note that when using multiple SMs (for redundancy), this option should NEVER be used.
-l <LMC> --lmc <LMC>	This option specifies the subnet LMC value. The number of LIDs assigned to each port is 2^LMC. The LMC value must be in the range 0-7. LMC values > 0 allow multiple paths between ports. Default: OpenSM defaults to LMC = 0, which allows one path between any two ports.	Use when "path migration" option is required. Note that in order to take advantage of path migration, the Connection Manager (or any other method of RC communication setup) should take additional steps. LMC values > 0 should only be used if the subnet topology provides multiple paths between ports, i.e. multiple interconnects exist between switches.
-i <eq-ignore-guids-file> -ignore-guids <eq-ignore-guids-file>	This option provides means to define a set of ports (by guids) that will be ignored by the link load equalization algorithm.	If there are some ports on the fabric that are rarely used (like a dedicated OpenSM node), it is possible to specify their guids. This way, their BW will be ignored by the routing algorithm.
-d <number> --debug <number>	This option specifies debug behavior. The number following -d selects the debug option (can be specified multiple times): -d 0 - Ignore other SM nodes. -d 1 - Force single threaded dispatching. -d 2 - Force log flushing after each log message. -d 3 - Disable multicast support. -d 4 - Put OpenSM in memory tracking mode. -d 10. Put OpenSM in testability mode. Default: No debug options are enabled.	These options are not normally needed.
-u	This option activates the UPDN routing algorithm instead of the (default) Min Hop algorithm	When there is a deadlock (e.g., due to high pressure) in a loop of the subnet (refer to "UPDN Unicast Routing Algorithm" on page 31 for more details)
-a <guid_list_file>	This option is active only when the UPDN algorithm is activate (option -u). It specifies the guid list file in which each guid is specified on a separate line	When the user wishes to manually specify the nodes of the subnet to be used as roots of the UPDN algorithm

3.2 osmsh

osmsh (short for "OpenSM shell") is intended to provide an open customization solution for Subnet Management. To facilitate full customization and configuration of subnet management, it is required to define policies that control how the following tasks are performed:

- LID assignment
- Unicast Routing
- Partitioning of the subnet (similar to the VLAN concept)
- Setting of Service Levels and their relative arbitration rules.

Rather than defining a specific syntax for the definition of each policy, the user is allowed to write arbitrary algorithms which directly interact with the lower level interface controlling the fabric. Such algorithms can be implemented in TCL (or as shared C/C++ libraries). The algorithm may use an additional small set of ***osmsh*** commands as well as access the internal data model of the SM.

osmsh is built as a TCL package called ***osm*** which should be loaded into TCL. Once OpenSM is installed, the package is located in the lib directory.

> Note: If the lib directory where the package is installed is not in the default path, it must be added to TCLLIBPATH in order for TCL to allocate it.

To load the package enter:

```
prompt> tclsh8.3
% package require osm
osm >
```

The API and data model are described in the following sections:

3.2.1 *osmsh* options

osmsh provides full access to all ***opensm*** *internal* options which control various aspects of its operation. The available options are listed in the following table.

Table 2 - OpenSM / OsmSh exposed options

Option	Default and Units	Usage
m_key	0	MKey used by the SM Set(PortInfo)
sm_key	0	SMKey used by the SA to qualify a query as "trusted"

Table 2 - OpenSM / OsmSh exposed options

Option	Default and Units	Usage
subnet_prefix	0xf800000000000000	The subnet prefix to be used by SM/SA
m_key_lease_period	0	MKey lease period included in Set(PortInfo)
sweep_interval	10 sec	Interval between sweeps
max_wire_smps	1	Number of simultaneous SMPs on the wire
transaction_timeout	100 msec	The time between a request and its expected response
sm_priority	1	The priority of the SM with respect to other SMs
lmc	0	2^{lmc} is the number of LIDs assigned to each port
max_op_vls	1	The maximal number of operational VLs used
reassign_lids	FALSE	If true - new LIDs will be assigned
reassign_lfts	TRUE	If true - existing LFT values are ignored on first sweep
ignore_other_sm	FALSE	If true - no handoff compliancy.
single_thread	TRUE	If true - use a single thread for SMP processing.
no_multicast_option	FALSE	If true - no multicast support by SA ClassPortInfo.
disable_multicast	FALSE	If true - no multicast GSI support.
force_log_flush	TRUE	If true - force flush of the log file on every log.
subnet_timeout	18 dec	time=4us*$2^{subnet\_timeout}$. Used for Trap resend.
packet_life_time	20 dec	time=4us*$2^{plt\_timeout}$. Max life time for a packet on the switch. The default value turns off this mechanism.
head_of_queue_lifetime	20 dec	time=4us*$2^{hoq\_timeout}$. Max time for a packet at the head of the Tx queue. The default value turns off this mechanism.
local_phy_errors_threshold	8	The number of consecutive PHY errors that will cause a Trap.
overrun_errors_threshold	8	The number of buffer overrun errors that will cause a Trap.
polling_timeout	1000 msec	Time between polls of the other Master SM
polling_retry_number	4	Number of failing other Master SM polls that will cause re-discovery.
force_heavy_sweep	FALSE	If true - makes every sweep scan through the entire subnet.
sweep_on_trap	TRUE	Start a heavy sweep when trap is received
max_port_profile	XX	Deprecated - do not allow link over-subscription above this value.
port_profile_switch_nodes	FALSE	If true - will include switch nodes in the link subscription counting. Otherwise ignore them.

3.2.1.1 Setting Some Options

The ***osmsh*** options are exposed to TCL as a TCL object. As such this object provides the standard "cget" and "configure" methods known to the experienced TCL programmer. To get the list of available ***osmsh*** options, use the following command:

```
osm_opts configure
=> { -m_key -sm_key -subnet_prefix -m_key_lease_period -sweep_interval -max_wire_smps
-transaction_timeout -sm_priority -lmc -max_op_vls -reassign_lids -reassign_lfts -ignore_other_sm
-single_thread -no_multicast_option -disable_multicast -force_log_flush -subnet_timeout
-packet_life_time -head_of_queue_lifetime -local_phy_errors_threshold -overrun_errors_threshold
-polling_timeout -polling_retry_number -force_heavy_sweep -sweep_on_trap -max_port_profile
-port_profile_switch_nodes  }
```

To retrieve the value of an option use:

```
osm_opts cget <option>
=> <value>
```

Example:

```
osm_opts cget -sweep_interval
=>10
```

And finally setting an option:

```
osm_opts configure <option> <value>
=> <value>
```

Example:

```
osm_opts configure -sweep_interval 20
=>20
```

3.2.1.2 Setting Verbose Options

There is a special function used for setting the verbosity level. For that enter:

```
osm_set_verbosity_level <verbosity_level>
=>0
```

Where the verbosity level is defined by the following:

none - 0x00
errors - 0x01
information data - 0x02
verbose information - 0x04
debug information - 0x08
routing information - 0x40

These flags can be OR-ed, and used in parallel.
For example - for using partial verbosity (the parallel of running opensm -v):

```
osm_set_verbosity 0x07
=>0
```

For running with full verbosity (parallel to running opensm -V):

```
osm_set_verbosity 0xff
=>0
```

The verbosity level is set by default to 1.

To retrieve the value of the verbosity level use:
osm_opts cget -log_flags
=>7
(If we defined the verbosity as 0x07)

3.2.1.3 Setting The Log File

To set the log file enter:

```
osm_set_log_file <log_file_name>
=>0
```

The default log file used is /tmp/osm.log

Note that defining the log file should be done before initiating the ***osm*** object (that is, before running osm_init). Using osm_set_log_file after osm_init will not take effect and the default name prevails.

To retrieve the name of the log file enter:

```
osm_opts cget -log_file
=>/tmp/osm.log
```

(If no other log file was defined)

3.2.1.4 Activating the UPDN Routing Algorithm

The default routing algorithm is Min Hop. To activate the UPDN Unicast routing algorithm enter:

```
osm_opt configure -updn_activate TRUE
```

To set the guid file list, enter one of the following:

```
osm_updn_set_guid_file <updn guid file name>
```

or

```
osm_opts configure -updn_guid_file <updn guid file name>
```

Please refer to “UPDN Unicast Routing Algorithm” (page 31) for more details.

3.2.2 Building a Basic SM Flow (Hello World)

Any ***osmsh*** flow should include the following stages: initiating the ***osm*** object, binding ***osmsh*** to an IB port, starting a sweep, and shutting down ***osmsh***.

3.2.2.1 Initiating an *osm object*

To initiate an ***osm*** *object* enter:

```
osm_init
=> 0
```

3.2.2.2 Binding *osmsh* to an IB port

To bind ***osmsh*** to an IB port, you will need to find a local IB port which is not in the DOWN state. The following command allows you to inspect the status of all IB ports:

```
osm_get_local_ports_info
=> { {guid1 lid1 link-state1} {guid2 lid2 link-state2} ...}
```

Example:

```
osm_get_local_ports_info
=> {0x0002c901093d91c1 0x0400 DOWN} {0x0002c901093d91c2 0x0500 ACTIVE}
```

The first port reported (HCA port #1) is down: either no cable is connected to the port, or the other side of it is connected to a non-active device (power off or no driver).

The next step is to bind ***osmsh*** to a specific port using the following command:

```
osm_bind <port guid>
=> 0/1 0 = success, 1 = error
```

Example: (continued)

```
osm_bind 0x0002c901093d91c2
=> 0
```

3.2.2.3 Starting a sweep

To start a sweep enter:

```
osm_sweep
=> 0
```

Before progressing to inspecting the results, wait for the subnet to be initialized.

```
osm_wait_for_subnet_up
=> 0
```

3.2.2.4 Shutting down *osmsh*

To shut down ***osmsh*** and exit the program enter:

```
exit
```

This will destroy the ***osm*** object, clean all data models and exit TCL.

3.2.2.5 A Complete Basic Flow

To complete the basic flow, a routine to find the first active port is provided. The following is an example of a full flow that can be saved to a file and run as is. (The first part of this program is an example of the routine mentioned above).

```
#!/bin/sh
# the next line restarts using tclsh \
    exec tclsh8.3 "$0" $@"

# First - load the osm package
package require osm

# Find the first available port that is not DOWN and
# return its GUID
proc OSM_GetFirstAvailablePortGuid {} {
  foreach GuidLidStat [osm_get_local_ports_info] {
     set portState [lindex $GuidLidStat 2]
     if {$portState != "DOWN"} {
        return [lindex $GuidLidStat 0]
     }
  }
  return ""
}
```

```
# Initiate the osm object
osm_init

# Use the first available port (active or init)
set port_guid [OSM_GetFirstAvailablePortGuid]
if {$port_guid == ""} {
  puts "-E- Fail to find any available port"
  exit
}
puts "-I- Attaching OpenSM to Port: $port_guid"
if {[osm_bind $port_guid]} {
  puts "-E- Fail to bind OSM to $port_guid"
  exit
}
# SWEEP IT
osm_sweep

# Better to wait for it then miss something  ...
osm_wait_for_subnet_up

# Loop forever until shut down
while {! $osm_exit_flag} {after 5000}

# Exit the program
exit
```

3.2.3 Interactive Flow

Several commands are provided to enable running the sweep in a step by step manner. It is recommended to turn off the sweep_on_trap option when running interactively. Otherwise, any trap received will trigger a new sweep, thus complicating the task of tracking the program state.

Table 3 - Manual and Interactive Sweep

Synopsis	Usage
osm_start_sweep	Initiates a new complete sweep immediately. Note that requesting another sweep in the middle of a running one will be ignored silently.
osm_run_discover	Run the first single step: Discover the fabric.
osm_run_lid_assign	Run the second single step. Must be called right after the discovery step.
osm_run_calc_lfdb	Run the third single step: Calculate the min-hop-count tables.
osm_run_set_lfdbs	Assign routes by setting the LFTs of all the switches
osm_run_set_mfdbs	Calculate and Set the Multicast Forwarding Tables.
osm_run_arm	Bring all links to ARM state (if are in INIT).
osm_run_activate	Bring all links to ACTIVE state.
osm_run_continuous	Resume continuous (non-single step) sweep mode.

If you plan to use interactive mode - it is highly recommended to install the Tcl extension named: tclreadline. This will provide you superior interactive features in ***osmsh***: command completion, editing, history and log.

3.2.4 Data Model

osmsh provides access to some of the internal data structures of the SM in the form of object identifiers. A set of access functions can be used to obtain these identifiers. The access functions and the format of the identifiers are described in the following table.

Table 4 - Exposed objects and their access functions

Object Type	Class	Access Function	Identifier Format
Nodes	osm_node_t	osm_get_nodes	node:<node-guid>
Ports	osm_port_t	osm_get_ports	port:<port-guid>
Physical Ports	osm_physp_t	osm_get_physp <node> <num>	physp:<node-guid>_<port_num>
Switches	osm_switch_t	osm_get_switches	switch:<node-guid>

Example of Usage: A Simple Report Line

```
puts "OSMSH: Discovered [llength [osm_get_nodes]] nodes [llength [osm_get_switches]] switches"
```

osmsh objects are standard "Swig-Tcl" objects. As such they have two flavors for their usage: identifiers and objects.

3.2.4.1 Accessing the Data Model Using Identifiers

As described above, the identifiers can be obtained by using access functions. Once an object identifier is stored in a variable, each one of its attributes can be obtained by using a "get" method, or modified using a "set" method. The format of the get/set methods is: <class>_<attribute>_<get|set>

For Example:

```
# obtain all discovered nodes
set nodes [osm_get_nodes]
# get the first node
set node  [lindex $nodes 0]
# get the node description
osm_node_t_node_desc_get $node
```

3.2.4.2 Accessing Objects

Any identifier can be converted into a Tcl "Object". As such, like any other Tcl object, it provides the standard "cget" and "configure" methods for inspecting and modifying its attributes.

In order to convert an identifier into an object use the following command:

```
<class> <obj_name> -this <obj identifier>
```

Example (using the previous identifier stored in $node):

```
osm_node_t myNodeObj -this $node
```

Once declared, the <obj_name> can be used in conjunction with the standard "configure" and "cget" commands.

Example (following the previous one):

```
myNodeObj cget -node_desc
=> MT47396 Infiniscale-iii Mellanox Technologies
```

3.2.4.3 Objects Content Info

The following code can be loaded into ***osmsh*** and used to dump out the content of every identifier or object.

```
proc objDump {obj} {
  catch {$obj cget} atts
  puts "---- Object Dump ------"
  foreach attr [lindex $atts 0] {
    set an [string range $attr 1 end]
    puts "$an = [$obj cget $attr]"
  }
  puts "-----------------------"
}

proc identifierDump {class id} {
  if {[catch {$class __obj -this $id} e]} {
    puts $e
  } else {
    objDump __obj
    rename __obj ""
  }
}

set node [lindex [osm_get_nodes] 0]
=> node:0x0002c90120267d40
identifierDump osm_node_t $node
=>---- Object Dump ------
this = node:0x0002c90120267d40
node_info = {base_version 1} {class_version 1} {node_type 2} {num_ports 8} {sys_guid 0x0000000000000000} {node_guid
0x0002c90120267d40} {port_guid 0x0002c90120267d40} {partition_cap 32} {device_id 43132} {revision 160} {port_num_vendor_id
50332361 }
node_desc = MT43132 Mellanox Technologies
discovery_count = 1
physp_tbl_size = 9
-----------------------



osm_node_t myNode -this $node
objDump myNode
=>---- Object Dump ------
this = node:0x0002c90120267d40
node_info = {base_version 1} {class_version 1} {node_type 2} {num_ports 8} {sys_guid 0x0000000000000000} {node_guid
0x0002c90120267d40} {port_guid 0x0002c90120267d40} {partition_cap 32} {device_id 43132} {revision 160} {port_num_vendor_id
50332361 }
node_desc = MT43132 Mellanox Technologies
discovery_count = 1
physp_tbl_size = 9
-----------------------
```

3.2.5 LID assignment Policy

Real world fabrics require "hot-plug" support, i.e., it should be possible for two disjoint subnets to be unified (by simply connecting two switches). If the two disjoint subnets have overlapping LID assignments, these LID collisions must be resolved. Reassigning the LIDs will result in the loss of packets, and major re-configuration of the Unicast forwarding tables (LFT) will be needed.

To avoid such re-configuration havoc, ***osmsh*** supports a method by which one can pre-assign LIDs to specific IB devices. This pre-assignment is referred to as "LID assignment policy". One way to perform this is to base the assignment on a static list of GUIDs: by this, it is possible to statically pre-assign each existing GUID to a LID.

Since ***osmsh*** (and ***opensm***) assigns LIDs only after completing a full discovery of the subnet, the pre-assignment should be performed after the ***osmsh*** discovery stage. To fulfill this requirement, ***osmsh*** allows registering a user defined procedure or Tcl expression to be executed between the discovery stage and the assignment of LIDS. This is accomplished by using the following command:

```
osm_reg_pre_lid_assign_cmd <tcl expression or procedure name>
```

Finally, the command provided for associating a LID with a Port GUID is:

```
osm_physp_set_lid <portId> <lid>
where:
portId - is a physical port identifier returned by osmsh
```

The code on the following page provides a complete example demonstrating how this "LID assignment policy" feature can be used in practice.

```
# PROC: assoc <key> <key value list>
# given a key and a list of ley/value pairs get the pair
proc assoc {key key_list} {
  foreach kv $key_list { if {[lindex $kv 0] == $key} {return [lrange $kv 1 end]}  }
  return ""
}

# PROC: OSM_PreLidAssign
# this routine will pre assign lids based on a global variable mapping of port guid to lid: GUID_TO_LID_TBL(guid) -> lid
proc OSM_PreLidAssign {} {
  global GUID_TO_LID_TBL
  # go over all nodes
  foreach node [osm_get_nodes] {
    # need to decide if it will have a single guid or multiple:
    set nodeInfo [osm_node_t_node_info_get $node]
    set nodeType [assoc node_type $nodeInfo]
    set numPorts [assoc num_ports $nodeInfo]
    # if not a switch we can assign lids to each port
    if {$nodeType != 2} {
      # go over all physical ports available and set lid
      for {set pn 1} { $pn <= $numPorts } { incr pn } {
        # might be invalid phys port
        if {[catch {set port [osm_get_physp $node $pn]}]} { continue }
        set port_guid [osm_physp_t_port_guid_get $port]
        if {[info exists GUID_TO_LID_TBL($port_guid)]} {
          set lid $GUID_TO_LID_TBL($port_guid)
          catch {osm_physp_set_lid $port $lid}
          puts "OSM_PreLidAssign Setting Port:$port_guid Lid:$lid"
        } else {
          puts "OSM_PreLidAssign Ignoring undef port:$port guid:$port_guid"
        }
      }
    } else {
      # might be invalid
      if {[catch {set port [osm_get_physp $node 0]}]} {continue}
      # only first port if switch
      set port_guid [osm_physp_t_port_guid_get $port]
      # switch node - only port 0 requires a lid
      if {[info exists GUID_TO_LID_TBL($port_guid)]} {
        set lid $GUID_TO_LID_TBL($port_guid)
        osm_physp_set_lid $port $lid
        puts "OSM_PreLidAssign Setting Port:$port_guid Lid:$lid"
      } else {
        puts "OSM_PreLidAssign Ignoring undef sw port:$port guid:$port_guid"
      }
    }
  }
}

# finally register the function to be invoked during the sweeps:
osm_reg_pre_lid_assign_cmd OSM_PreLidAssign
```

3.2.6 Routing Policy - Integrating a User's Routing Engine

The ***osmsh*** routing algorithm basically equalizes the link load between all the links with the minimal hop count to the target LID. Being a good generic algorithm, it might be adequate for many IB fabrics, however, it is usually not optimal for configurations with non-heterogeneous endnodes. To provide the means to load or calculate a better Unicast routing scheme, ***osmsh*** supports registering a user defined Tcl expression to be evaluated before the internal algorithm for Unicast Linear Forwarding Table assignment is invoked. A simple interface for assigning a single LFT entry is also provided.

Note that for complex dynamic cases where the topology frequently changes, an online router might be needed. Tcl is not the most adequate language for implementing this algorithm; rather, it is better to code it in C/C++ and dynamically load a shared object into ***osmsh***. The Tcl should only be used to invoke the main routing task.

The following command assigns an LFT entry for a given LID on a given switch:

```
osm_switch_fdb_set <switchId> <lid> <port_num>
switchId - is a switch identifier (as returned by osm_get_switches).
```

The following command registers the given Tcl command to be invoked before the Unicast LFT assignment:

```
osm_reg_ucast_fdb_assign_cmd <tcl expression or procedure name>
```

The code below provides an example of how to use these interfaces for loading pre-calculated routing:

```
# This proc will use the preloaded FDB data and pre-assign them.
# It assumes a global variable holding a map of switch guid to list of LID, PORT pairs
# for each switch
proc OSM_SetSwitchesFDBs {} {
  global OSM_SW_FDB

  # track number of pre-assignments
  set num 0

  # go over all switches and check if their we have pre-assigned FDB
  foreach sw [osm_get_switches] {
    # get the sw guid
    set guid [string range $sw 7 end]

    # ok we got some FDB entries to set
    if {[info exists OSM_SW_FDB($guid)]} {
      foreach {lid port_num} $OSM_SW_FDB($guid) {
        osm_switch_fdb_set $sw $lid $port_num
        incr num
      }
    }
  }
  puts "OSM_SetSwitchesFDBs Pre-Assigned $num FDB entries"
}
osm_reg_ucast_fdb_assign_cmd OSM_SetSwitchesFDBs
```

3.2.7 Multicast Routing Policy - Integrating A User's Multicast Routing Engine

The ***osmsh*** multicast routing algorithm basically creates a minimum hop tree to all ports in the multicast group. Being a good generic algorithm, it might be adequate for many IB fabrics, however, it is usually not optimal for some configurations. To provide the means to load or calculate a better Multicast routing scheme, ***osmsh*** supports registering a user defined Tcl expression to be evaluated during calls for multicast create/join/leave requests. A simple interface for assigning a single MFT entry is also provided.

Note that for complex dynamic cases where the topology frequently changes, an online router might be needed. Tcl is not the most adequate language for implementing this algorithm; rather, it is better to code it in C/C++ and dynamically load a shared object into ***osmsh***. The Tcl should only be used to invoke the main routing task.

The following two commands assign an MFT entry for a given MLID on a given switch:

1. To add a specific port to the MLID entry use:

osm_switch_mcfdb_set_port_num <switched> <mlid> <port_num>

switchId - is a switch identifier (as returned by osm_get_switches).

2. To define the entire entry for the given MLID with the given ports list use:

osm_switch_mcfdb_set <switched> <mlid> <ports_list>

switchId - is a switch identifier (as returned by osm_get_switches).

ports_list - is a list of ports to be added (e.g: "1 2 7" will define for this switch mlid entry only ports 1,2 &7 and clean all the rest, ""will clean the entry for this mlid on the given switch).

The following command registers the given Tcl command to be invoked on every call for multicast create/join/leave or subnet change:

osm_reg_mcast_fdb_assign_cmd <tcl procedure name>

The Tcl procedure will be called with the following variables:

procedure_name <mlid> <multicast_call_type> <port_guid>

multicast_call_type - is a string defining the cause for this call. Possible values are:

OSM_MCAST_REQ_TYPE_CREATE - call was invoked due to a call for creation of multicast group.

OSM_MCAST_REQ_TYPE_JOIN - call was invoked due to a call for joining a port to a multicast group.

OSM_MCAST_REQ_TYPE_LEAVE - call was invoked due to a call for removing a port from a multicast group.

OSM_MCAST_REQ_TYPE_SUBNET_CHANGE - call was invoked due to some change in the subnet that might cause a change in the multicast tree.

port_guid - is the guid of the port being added to the multicast group (in CREATE/JOIN cases) or removed (in LEAVE cases). If the call was with OSM_MCAST_REQ_TYPE_SUBNET_CHANGE then port_guid is zero.

Note that during the creation of a new multicast group or the deletion of a multicast group, all the relevant multicast entries for all switches are cleaned.

The code below provides an example of how to use these interfaces for loading pre-calculated multicast routing:

```
# This proc will use the preloaded MCFDB data and assign them.
# Should be registered using osm_reg_mcast_fdb_assign_cmd
# The function will add the mlid entry of the switch when creating
# the mc group or when joining a port or when there is some
# change in the subnet.
# leave commands to the mc group will be ignored.
```

```
proc OSM_SetSwitchesMCFDBs {mlid req_type port_guid} {
global OSM_SW_MCFDB

  set num 0
  puts "SetSwitchesMCFDBs Enter set switches callback function"
  if {$req_type == "OSM_MCAST_REQ_TYPE_CREATE" || \
       $req_type == "OSM_MCAST_REQ_TYPE_JOIN" || \
       $req_type == "OSM_MCAST_REQ_TYPE_SUBNET_CHANGE" } {
    # go over all switches and check if we have pre-assigned MCFDB
    foreach sw [osm_get_switches] {
      # get the sw guid
      set guid [string range $sw 7 end]
      # ok we got some MCFDB entries to set
      if {[info exists OSM_SW_MCFDB($guid)]} {
        foreach lidNports_list $OSM_SW_MCFDB($guid) {
          set lid [lindex $lidNports_list 0]
          set ports_list [lindex $lidNports_list 1]
          if {$lid == $mlid} {
            osm_switch_mcfdb_set $sw $lid $ports_list
          }
        }
incr num
      }
    }
} elseif {$req_type == "OSM_MCAST_REQ_TYPE_LEAVE"} {
    puts "OSM_SetSwitchesMCFDBs: Ignore LEAVE command."
  }
puts "OSM_SetSwitchesMCFDBs Pre-Assigned $num MCFDB entries"
}
osm_reg_mcast_fdb_assign_cmd OSM_SetSwitchesMCFDBs
```

3.2.8 Configuring Quality of Service

osmsh provides only rudimentary support for configuring Quality of Service. As Quality of Service involves setting a device table of three different types (SL2VL map, VL Arbitration map and PortInfo), ***osmsh*** provides the means to set these attributes and send them over to remote devices. To achieve this, ***osmsh*** pre-allocates a single Tcl object containing each of these types. The object provides the standard “cget” and “configure” methods by which the user can inspect and modify the content of an attribute. Two additional methods are provided: (1) “send” allows sending the attribute to a remote device; (2) “clear” resets the attribute value to zero.

Note that the response for the “send” method (which translates into a standard SubnAdm.Set MAD) is a standard Get-Response MAD. As such ***osmsh*** will accept this response and update the internal database with the Set result, so the algorithm can verify the Set was successful.

The Objects listed in the following table are pre-allocated:

Table 5 - Pre-Allocated MAD Objects Required for QoS Flows

Object	Send method Synopsis	Send Parameters
osm_port_info_mad	<physp-id>	physp-id = phys port identifier (as obtained from osm_get_physp)
osm_slvl_tbl_mad	<physp-id> <in-port-#>	physp-id = phys port identifier; in-port-# = input port number
osm_vl_arb_tbl_mad	<physp-id> <block-#>	physp-id = phys port identifier; block-# = the VLArb table block

The following is an example demonstrating how to use these objects:

```
# get the current value of the VL Arbitration table object.
set vl [osm_vl_arb_tbl_mad cget -vl_entry]
# replace some entries
set vl [lreplace $vl 0 0 {1 100}]
# set the VL Arb object with your changes
osm_vl_arb_tbl_mad configure -vl_entry $vl
# get a particular node
set nodes [lindex [osm_get_nodes] 0]
# get a phys port to work with (we use port #1)
set phys [osm_get_physp $node 1]
# now send the VL Arb to this physical port.
osm_vl_arb_tbl_mad send $phys 1
# verify the result by query of the phys port VL Arb table.
# NOTE: you need to wait for the SubnAdm.GetResp(VLArbTable) to arrive
# wait_time > retry-count*subnet timeout+processing-time
osm_physp_t_vl_arb_get $phys
```

3.2.9 Configuring Partitions (VLANs)

InfiniBand provides extensive support for partitioning a large fabric into smaller (sometimes overlapping) partitions. It does that by specification of hardware-assisted packet filtering, based on a packet’s source partition at both input and output ports of every IB device. A port is part of a partition if the PKey (Partition Key) is stored in its Partition Key Table. Thus, ***osmsh*** provides the means to modify partition table contents, and set the appropriate fields in the

PortInfo and NodeInfo that control partition-based packet filtering. The mechanism provided is exactly the same as the one described in the previous section, and includes support for the following MADs:

Table 6 - Pre-Allocated MAD Objects Required for Partitioning

Object	Send method Synopsis	Send Parameters
osm_port_info_mad	<physp-id>	physp-id = phys port identifier (as obtained from osm_get_physp)
osm_node_info_mad	<node-id>	node-id = node identifier (as obtained from osm_get_nodes)
osm_pkey_tbl_mad	<physp-id> <block-#>	physp-id = phys port identifier; block-# = the PKey table block

3.3 OsmTest

OsmTest provides a test suite for OpenSM and OsmSh. Its executable is invoked by typing ***osmtest***.

OsmTest has the following capabilities and testing flows:

- It creates an inventory file of all available Nodes, Ports, and PathRecords, including all their fields.
- It verifies the existing inventory, with all the object fields, and matches it to a pre-saved one.
- A Multicast Compliancy test.
- An Event Forwarding test.
- A Service Record registration test.
- An RMPP stress test.
- A Small SA Queries stress test.

It is recommended that after installing OpenSM, the user should run "***osmtest*** -f c" to generate the inventory file, and immediately afterwards run "***osmtest*** -f a" to test OpenSM.

Another recommendation for OsmTest usage is to create the inventory when the IB fabric is stable, and occasionally run "osmtest -v" to verify that nothing has changed.

The following table provides a full description of all ***osmtest*** options.

Table 7 - OsmTest Command Line Options.

Option	Effect
-g <GUID in hex> --guid <GUID in hex>	OsmTest will bind to the port with the provided GUID. Default is to present to user the available GUIDs, and to select one of them. Use this option to avoid the interaction required in order to select the port. It is possible to enter: "echo 1 | opensm" to select the first GUID.
-i <filename> --inventory <filename>	This option specifies the name of the inventory file. Normally, ***osmtest*** expects to find an inventory file with which it validates real-time information received from the SA during testing. Default: ***osmtest*** uses the file 'osmtest.dat'.
-f <c|a|v|s|e|f|m|q> --flow <c|a|v|s|e|f|m|q>	The actual test flow run by OsmTest: c = create an inventory file with all nodes, ports and paths. a = run all validation tests (expects an input inventory) v = only validate the given inventory file. s = run service registration, un-registration and lease. e = run event forwarding test. f = flood the SA with queries according to the stress mode. m = multicast flow. q = QoS info - Dump VLArb and SLtoVL tables. Default flow: all the above but QoS
-s <level> --stress <level>	This option runs the specified stress test instead of the normal test suite. Stress test options are: -s 1 - Single-MAD response SA queries -s 2 - Multi-MAD (RMPP) response SA queries. Default: stress testing is not performed.
-t <milliseconds> --timeout <milliseconds>	This option specifies the time in milliseconds used for transaction timeouts (request to response). Default: 100ms.
-v --verbose	This option increases the log verbosity level. The -v option may be specified multiple times to further increase the verbosity level.

Table 7 - OsmTest Command Line Options.

Option	Effect
-V	This option sets the verbosity level to the maximum and forces log flushing.
-l <file_name> --log_file <file_name>	This option is used to specify the log file name. By default the log goes to standard output.
-d <number> --debug <number>	This option specifies debug behavior. The number following -d selects the debug option to be enabled (can be specified multiple times): -d 0 - Unused. -d 1 - Do not scan and compare Path Record (should be used on large clusters as #Paths = Nodes^2) -d 2 - Force log flushing after each log message. -d 3 - Unused. Default: no debug options are enabled.

4 UPDN Unicast Routing Algorithm

OpenSM offers two routing engines:

1. Min Hop Algorithm - based on the minimum hops to each node where the path length is optimized.
2. UPDN Unicast routing algorithm - also based on the minimum hops to each node, but it is constrained to ranking rules. This algorithm should be chosen if the subnet is not a pure Fat Tree, and a deadlock may occur due to a loop in the subnet.

The UPDN algorithm is installed as part of the OpenSM package and is placed under the lib directories. The algorithm is described in the following sections:

- "UPDN Algorithm Purpose" (page 31)
- "UPDN Algorithm Usage" (page 32)

4.1 UPDN Algorithm Purpose

The UPDN algorithm is designed to prevent deadlocks from occurring in loops of the subnet. A loop-deadlock is a situation in which it is no longer possible to send data between any two hosts connected through the loop. As such, the UPDN routing algorithm should be used if the subnet is not a pure Fat Tree, and one of its loops may experience a deadlock (due, for example, to high pressure).[1]

The UPDN algorithm is based on the following main stages:

1. Auto-detect root nodes - based on the HCA hop length from any switch in the subnet, a statistical histogram is built for each switch (hop num vs number of occurrences). If the histogram reflects a specific column (higher than others) for a certain node, then it is marked as a root node. Since the algorithm is statistical, it may not find any root nodes. The list of the root nodes found by this auto-detect stage is used by the ranking process stage.

   Note 1: The user can override the node list manually (see Section 4.2).

   Note 2: If this stage cannot find any root nodes, and the user did not specify a guid list file, OpenSM defaults back to the Min Hop routing algorithm.

2. Ranking process - All root switch nodes (found in stage 1) are assigned a rank of 0. Using the BFS algorithm, the rest of the switch nodes in the subnet are ranked incrementally. This ranking aids in the process of enforcing rules that ensure loop-free paths.
3. Min Hop Table setting - after ranking is done, a BFS algorithm is run from each (HCA or switch) node in the subnet. During the BFS process, the FDB table of each switch node traversed by BFS is updated, in reference to the starting node, based on the ranking rules and guid values.

At the end of the process, the updated FDB tables ensure loop-free paths through the subnet.

1. To learn more about deadlock-free routing, see the article *"Deadlock Free Message Routing in Multiprocessor Interconnection Networks"* by William J Dally and Charles L Seitz (1985).

4.2 UPDN Algorithm Usage

4.2.1 Activation through *OpenSM*

Use '-u' for activating the UPDN algorithm

Use '-a <guid_list_file>' for adding an UPDN guid file that contains the root nodes for ranking.

If the '-a' option is not used, OpenSM uses its auto-detect root nodes algorithm.

See notes at the end of the following section.

4.2.2 Activating through *osmsh*

```
osm_opt configure -updn_activate TRUE
```

To add a guid file list, enter one of two:

```
osm_updn_set_guid_file <guid file name>
```

or

```
osm_opt configure -updn_guid_file <guid file name>
```

If no guid list file is specified, OpenSM uses its auto-detect root nodes algorithm.

Notes on the guid list file:

1. A valid guid file specifies one guid in each line. Lines with an invalid format will be discarded.
2. The user should specify the root switch guids. However, it is also possible to specify HCA guids; OpenSM will use the guid of the switch (if it exists) that connects the HCA to the subnet as a root node.

Revision History

Table 1 - Revision History Table

Date	Revision	Description
Jan 2005	1.11	Minor changes in installation flags
Dec 2004	1.10	Added the UPDN algorithm for Unicast routing description
Sep 2004	1.00	osmsh is now described as a tcl package (previously as an executable)
		Added description for the user's interface for a multicast routing engine under osmsh
Jun 2004	0.31	Fixed typo (script name is install.sh)
Jun 2004	0.30	Created the first revision