CLI and Scripting Reference

ACM7000 Remote Site Gateway
ACM7000-L Resilience Gateway
IM7200 Infrastructure Manager
CM7100 Console Servers

Version 2.0 Rev 4 2023-12-15

Safety

Please take care to follow the safety precautions below when installing and operating the console server:

- Do not remove the metal covers. There are no operator serviceable components inside. Opening or removing the cover may expose you to dangerous voltage which may cause fire or electric shock. Refer all service to Opengear qualified personnel.
- To avoid electric shock the power cord protective grounding conductor must be connected through to ground.
- Always pull on the plug, not the cable, when disconnecting the power cord from the socket.

Do not connect or disconnect the console server during an electrical storm. Use a surge suppressor or UPS to protect the equipment from transients.

FCC Warning Statement

This device complies with Part 15 of the FCC rules. Operation of this device is subject to the following conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference that may cause undesired operation.

Proper back-up systems and necessary safety devices should be utilized to protect against injury, death or property damage due to system failure. Such protection is the responsibility of the user.
This console server device is not approved for use as a life-support or medical system.
Any changes or modifications made to this console server device without the explicit approval or consent of Opengear will void Opengear of any liability or responsibility of injury or loss caused by any malfunction.
This equipment is for indoor use and all the communication wirings are limited to inside of the building.

Copyright

TABLE OF CONTENTS

CONFIGURATION FROM THE COMMAND LINE

For those who prefer to configure their *console server* at the Linux command line level (rather than use a browser and the Management Console), this chapter describes using command line access and the ***config*** tool to manage the *console server* and configure the ports etc.

This *config* documentation in this chapter walks thru command line configuration to deliver the functions provided otherwise using the Management Console GUI.

For advanced and custom configurations and for details using other tools and commands refer to the next chapter

When displaying a command, the convention used in the rest of this chapter is to use single quotes (') for user defined values (e.g. descriptions and names). Element values without single quotes must be typed exactly as shown.

After the initial section on accessing the *config* command the menu items in this document follow the same structure as the menu items in the web GUI.

1.1 Accessing *config* from the command line

The *console server* runs a standard Linux kernel and embeds a suite of open source applications. So if you do not want to use a browser and the Management Console tools, you are free to configure the *console server* and to manage connected devices from the command line using standard Linux and Busybox commands and applications such as *ifconfig, gettyd, stty, powerman, nut etc*. However these configurations may not withstand a *power-cycle-reset* or *reconfigure*.

Opengear provides a number of custom command line utilities and scripts to make it simple to configure the *console server* and ensure the changes are stored in the *console server*'s flash memory etc.

In particular the **config** utility allows manipulation of the system configuration from the command line. With *config* a new configuration can be activated by running the relevant configurator, which performs the action necessary to make the configuration changes live.

To access *config* from the command line:

- Power up the *console server* and connect the “terminal” device:
    - If you are connecting using the serial line, plug a serial cable between the *console server* local DB-9 console port and terminal device. Configure the serial connection of the terminal device you are using to 115200bps, 8 data bits, no parity and one stop bit
    - If you are connecting over the LAN then you will need to interconnect the Ethernet ports and direct your terminal emulator program to the IP address of the *console server* (192.168.0.1 by default)
- Log on to the *console server* by pressing ‘return’ a few times. The *console server* will request a username and password. Enter the username *root* and the password *default*. You should now see the command line prompt which is a hash (#)

The *config* tool

Syntax

config [-ahv] [-d id] [-g id] [-p path] [-r configurator] [-s id=value] [-P id]

Description

The *config* tool is designed to perform multiple actions from one command if need be, so if necessary options can be chained together.

The *config* tool allows manipulation and querying of the system configuration from the command line. Using *config* the new configuration can be activated by running the relevant *configurator* which performs the action necessary to make the configuration changes live.

The custom user configuration is saved in the */etc/config/config.xml* file. This file is transparently accessed and edited when configuring the device using the Management Console browser GUI. Only the user *'root'* can configure from the shell.

By default, the *config* elements are separated by a '.' character. The root of the *config* tree is called <*config*>. To address a specific element place a '.' between each node/branch e.g. to access and display the description of *user1* type:

 # config -g config.users.user1.description

The root node of the *config* tree is <config>. To display the entire *config* tree, type:

 # config -g config

To display the help text for the *config* command, type:

 # config -h

The *config* application resides in the */bin* directory. The environmental variable called *PATH* contains a route to the */bin* directory. This allows a user to simply type *config* at the command prompt instead of the full path */bin/config*.

Options

-a –run-all	Run all registered configurators. This performs every configuration synchronization action pushing all changes to the live system
-x	Display the network hosts, local services and ports for the config command in XML
-h –help	Display a brief usage message
-v –verbose	Log extra debug information
-d –del=id	Remove the given configuration element specified by a '.' separated identifier
-g –get=id	Display the value of a configuration element
-p –path=file	Specify an alternate configuration file to use. The default file is located at */etc/config/config.xml*
-r –run=configurator	Run the specified registered configurator. Registered configurators are listed below.
-s --set=id=value	Change the value of configuration element specified by a '.' separated identifier
-e --export=file	Save active configuration to file
-i --import=file	Load configuration from file
-t --test-import=file	Pretend to load configuration from file
-S --separator=char	The pattern to separate fields with, default is '.'
-P --password=id	Prompt user for a value. Hash the value, then save it in id

The registered configurators are:

alerts
auth
cascade
console
dhcp
dialin
eventlog
hosts
ipaccess
ipconfig
nagios
power
serialconfig
services
secondary
systemsettings
time
ups
users

There are three ways to delete a config element value. The simplest way is use the *delete-node* script detailed later in Chapter 2. You can also assign the config element to "", or delete the entire config node using *-d*:

 # /bin/config -d 'element name'

All passwords are saved in plaintext *except* the user passwords and the system passwords, which are encrypted.

Note: The *config* command does not verify whether the nodes edited/added by the user are valid. This means that any node may be added to the tree. If a user were to run the following command:

 # /bin/config -s config.fruit.apple=sweet

The configurator will not complain, but this command is clearly useless. When the configurators are run (to turn the config.xml file into live config) they will simply ignore this <fruit> node. *Administrators* must make sure of the spelling when typing config commands. Incorrect spelling for a node will not be flagged.

Most configurations made to the XML file will be immediately active. To make sure that *all* configuration changes are active, especially when editing user passwords, run all the configurators:

 # /bin/config -a

For information on backing up and restoring the configuration file refer *Chapter 2 - Advanced Configuration*.

1.1.1 Serial Port configuration

The first set of configurations that needs to be made to any serial port are the RS232 common settings. For example to setup serial port 5 to use the following properties:

Baud Rate	*9600*	
Parity	*None*	
Data Bits	*8*	
Stop Bits	*1*	
label		*Myport*
log level		*0*
protocol	*RS232*	
flow control		*None*

To do this use the following commands:

 # config -s config.ports.port5.speed=9600
 # config -s config.ports.port5.parity=None

 # config -s config.ports.port5.charsize=8
 # config -s config.ports.port5.stop=1
 # config -s config.ports.port5.label=myport
 # config -s config.ports.port5.loglevel=0
 # config -s config.ports.port5.protocol=RS232
 # config -s config.ports.port5.flowcontrol=None

The following command will synchronize the live system with the new configuration:

 # config -r serialconfig

Note: Supported serial port baud-rates are ‘50’, ‘75’, ‘110’, ‘134’, ‘150’, ‘200’, ‘300’, ‘600’, ‘1200’, ‘1800’, ‘2400’, ‘4800’, ‘9600’, '19200', '38400', '57600', '115200', and '230400'.
Supported parity values are 'None', 'Odd', 'Even', 'Mark' and 'Space'.
Supported data-bits values are '8', '7', '6' and '5'.
Supported stop-bits values are '1', '1.5' and '2'.
Supported flow-control values are 'Hardware', 'Software' and 'None'.

Additionally, before any port can function properly, the mode of the port needs to be set. Any port can be set to run in one of the five possible modes: [*Console Server* mode | Device mode | Terminal server mode | Serial bridge mode]. All these modes are mutually exclusive.

***Console Server* mode**

The command to set the port in *portmanager* mode:

 # config -s config.ports.port5.mode=portmanager

To set the following optional config elements for this mode:

Data accumulation period	*100 ms*
Escape character	*% (default is ~)*
log level	*2 (default is 0)*
Shell power command menu	*Enabled*
RFC2217 access	*Enabled*
Limit port to 1 connection	*Enabled*
SSH access	*Enabled*
TCP access	*Enabled*
telnet access	*Disabled*
Unauthorized telnet access	*Disabled*

 # config -s config.ports.port5.delay=100
 # config -s config.ports.port5.escapechar=%
 # config -s config.ports.port5.loglevel=2
 # config -s config.ports.port5.powermenu=on
 # config -s config.ports.port5.rfc2217=on
 # config -s config.ports.port5.singleconn=on
 # config -s config.ports.port5.ssh=on
 # config -s config.ports.port5.tcp=on
 # config -d config.ports.port5.telnet
 # config -d config.ports.port5.unauthtel

Device Mode

For a device mode port, set the port type to either *ups, rpc,* or *enviro:*

 # config -s config.ports.port5.device.type=[ups | rpc | enviro]

For port 5 as a UPS port:

 # config -s config.ports.port5.mode=reserved

For port 5 as an RPC port:

 # config -s config.ports.port5.mode=powerman

For port 5 as an Environmental port:

 # config -s config.ports.port5.mode=reserved

Terminal server mode

Enable a TTY login for a local terminal attached to serial port 5:

 # config -s config.ports.port5.mode=terminal
 # config -s config.ports.port5.terminal=[vt220 | vt102 | vt100 | linux | ansi]

The default terminal is vt220

Serial bridge mode

Create a network connection to a remote serial port via RFC-2217 on port 5:

 # config -s config.ports.port5.mode=bridge

Optional configurations for the network address of RFC-2217 server of 192.168.3.3 and TCP port used by the RFC-2217 service = 2500:

 # config -s config.ports.port5.bridge.address=192.168.3.3
 # config -s config.ports.port5.bridge.port=2500

To enable RFC-2217 access: *# config -s config.ports.port5.bridge.rfc2217=on*

To redirect the serial bridge over an SSH tunnel to the server: *# config -s config.ports.port5.bridge.ssh.enabled=on*

Syslog settings

Additionally, the global system log settings can be set for any specific port, in any mode:

 # config -s config.ports.port#.syslog.facility='facility'
'facility' can be:

Default
local 0-7
auth
authpriv
cron
daemon
ftp
kern
lpr
mail
news

user
uucp
 # config -s config.ports.port#.syslog.priority='priority'
'priority' can be:
Default
warning
notice
Info
error
emergency
debug
critical
alert

1.1.2 Adding and removing Users

Firstly, determine the total number of existing Users (if you have no existing Users you can assume this is 0):

 # config -g config.users.total

This command should display *config.users.total 1*. Note that if you see *config.users.total* this means you have 0 Users configured.

Your new User will be the existing total plus 1. So if the previous command gave you 0 then you start with user number 1, if you already have 1 user your new user will be number 2 etc.

To add a user (with Username=John, Password=secret and Description =mySecondUser) issue the commands:

 # config -s config.users.total=2 (assuming we already have 1 user configured)
 # config -s config.users.user2.username=John
 # config -s config.users.user2.description=mySecondUser
 # config -P config.users.user2.password

NOTE: The -P parameter will prompt the user for a password, and encrypt it. In fact, the value of any config element can be encrypted using the -P parameter, but only encrypted user passwords and system passwords are supported. If any other element value were to be encrypted, the value will become inaccessible and will have to be re-set.

To add this user to specific groups (admin/users):

 # config -s config.users.user2.groups.group1='groupname'
 # config -s config.users.user2.groups.group2='groupname2'
etc...

To give this user access to a specific port:

 # config -s config.users.user2.port1=on
 # config -s config.users.user2.port2=on
 # config -s config.users.user2.port5=on
etc...

To remove port access:

 # config -s config.users.user2.port1='' (the value is left blank)
or simply:
 # config -d config.users.user2.port1

The port number can be anything from 1 to 48, depending on the available ports on the specific *console server*.

For example assume we have an RPC device connected to port 1 on the *console server* and the RPC is configured. To give this user access to RPC outlet number 3 on the RPC device, run the 2 commands below:

 # config -s config.ports.port1.power.outlet3.users.user2=John
 # config -s config.ports.port1.power.outlet3.users.total=2 (total number of users that have access to this outlet)

If more users are given access to this power outlet, then increment the *'config.ports.port1.power.outlet3.users.total'* element accordingly.

To give this user access to network host 5 (assuming the host is configured):

 # config -s config.sdt.hosts.host5.users.user1=John
 # config -s config.sdt.hosts.host5.users.total=1 (total number of users having access to host)

To give another user called 'Peter' access to the same host:

 # config -s config.sdt.hosts.host5.users.user2=Peter
 # config -s config.sdt.hosts.host5.users.total=2 (total number of users having access to host)

To edit any of the user element values, use the same approach as when adding user elements i.e. use the '-s' parameter. If any of the config elements do not exist, they will automatically be created.

To delete the user called John, use the delete-node script:

 # ./delete-node config.users.user2

The following command will synchronize the live system with the new configuration:

 # config -r users

1.1.3 Adding and removing user Groups

The *console server* is configured with a few default user groups (even though only two of these groups are visible in the Management Console GUI). To find out how many groups are already present:

 # config -g config.groups.total

Assume this value is six. Make sure to number any new groups you create from seven onwards.

To add a custom group to the configuration with Group name=Group7, Group description=MyGroup and Port access= 1,5 you'd issue the commands:

 # config -s config.groups.group7.name=Group7
 # config -s config.groups.group7.description=MyGroup
 # config -s config.groups.total=7
 # config -s config.groups.group7.port1=on
 # config -s config.groups.group7.port5=on

Assume we have an RPC device connected to port 1 on the console manager, and the RPC is configured. To give this group access to RPC outlet number 3 on the RPC device, run the two commands below:

 # config -s config.ports.port1.power.outlet3.groups.group1=Group7
 # config -s config.ports.port1.power.outlet3.groups.total=1 (total number of groups that have access to this outlet)

If more groups are given access to this power outlet, then increment the *'config.ports.port1.power.outlet3.groups.total'* element accordingly.

To give this group access to network host 5:

 # config -s config.sdt.hosts.host5.groups.group1=Group7
 # config -s config.sdt.hosts.host5.groups.total=1 (total number of groups having access to host)

To give another group called 'Group8' access to the same host:

 # config -s config.sdt.hosts.host5.groups.group2=Group8
 # config -s config.sdt.hosts.host5.groups.total=2 (total number of users having access to host)

To delete the group called Group7, use the following command:

 # rmuser Group7

Attention: The *rmuser* script is a generic script to remove any config element from config.xml correctly. However, any dependencies or references to this group will not be affected. Only the group details are deleted. The administrator is responsible for going through *config.xml* and removing group dependencies and references manually, specifically if the group had access to a host or RPC device.

The following command will synchronize the live system with the new configuration:

 # config -a

1.1.4 Authentication

To configure the Web Management Session Timeout:

 # config -s config.auth.sessionlifetime=5

To change the type of authentication for the *console server*:

 # config -s config.auth.type='authtype'

'authtype' can be:

Local
LocalTACACS
TACACS
TACACSLocal
TACACSDownLocal
LocalRADIUS
RADIUS
RADIUSLocal
RADIUSDownLocal
LocalLDAP
LDAP
LDAPLocal
LDAPDownLocal

To configure TACACS authentication:

 # config -s config.auth.tacacs.auth_server='comma separated list' (list of remote authentiction and authorization servers.)
 # config -s config.auth.tacacs.acct_server='comma separated list' (list of remote accounting servers. If unset, Authentication and Authorization Server Address will be used.)
 # config -s config.auth.tacacs.password='password'

To configure RADIUS authentication:

 # config -s config.auth.radius.auth_server='comma separated list' (list of remote authentiction and authorization servers.)
 # config -s config.auth.radius.acct_server='comma separated list' (list of remote accounting servers. If unset, Authentication and Authorization Server Address will be used.)
 # config -s config.auth.radius.password='password'

To configure LDAP authentication:

 # config -s config.auth.ldap.server='comma separated list' (list of remote servers.)
 # config -s config.auth.ldap.basedn='name' (The distinguished name of the search base. For example: dc=my-company,dc=com)
 # config -s config.auth.ldap.binddn='name' (The distinguished name to bind to the server with. The default is to bind anonymously.)
 # config -s config.auth.radius.password='password'

The following command will synchronize the live system with the new configuration:

 # config -r auth

1.1.5 Network Hosts

To determine the total number of currently configured hosts:

 # config -g config.sdt.hosts.total

Assume this value is equal to 3. If you add another host, make sure to increment the total number of hosts from 3 to 4:

 # config -s config.sdt.hosts.total=4

If the output is *config.sdt.hosts.total* then assume 0 hosts are configured.

Add power device host

To add a UPS/RPC network host with the following details:

IP address / DNS name	192.168.2.5
Host name	remoteUPS
Description	UPSroom3
Type	UPS
Allowed services	ssh port 22 and https port 443
Log level for services	0

Issue the commands below:

 # config -s config.sdt.hosts.host4.address=192.168.2.5
 # config -s config.sdt.hosts.host4.name=remoteUPS
 # config -s config.sdt.hosts.host4.description=UPSroom3
 # config -s config.sdt.hosts.host4.device.type=ups
 # config -s config.sdt.hosts.host4.tcpports.tcpport1=22
 # config -s config.sdt.hosts.host4.tcpports.tcpport1.loglevel=0
 # config -s config.sdt.hosts.host4.udpports.udpport2=443
 # config -s config.sdt.hosts.host4.udpports.udpport2.loglevel=0

The *loglevel* can have a value of 0 or 1.

The default services that should be configured are: *22/tcp (ssh), 23/tcp (telnet), 80/tcp (http), 443/tcp (https), 1494/tcp (ica), 3389/tcp (rdp), 5900/tcp (vnc)*

Add other network host

To add any other type of network host with the following details:

IP address / DNS name	192.168.3.10
Host name	OfficePC
Description	MyPC
Allowed sevices	ssh port 22,https port 443
log level for services	1

Issue the commands below. If the Host is not a PDU or UPS power device or a server with IPMI power control then leave the device type blank:

 # config -s config.sdt.hosts.host4.address=192.168.3.10
 # config -s config.sdt.hosts.host4.description=MyPC
 # config -s config.sdt.hosts.host4.name=OfficePC
 # config -s config.sdt.hosts.host4.device.type=" (leave this value blank)
 # config -s config.sdt.hosts.host4.tcpports.tcpport1=22
 # config -s config.sdt.hosts.host4.tcpports.tcpport1.loglevel=1
 # config -s config.sdt.hosts.host4.udpports.tcppport2=443
 # config -s config.sdt.hosts.host4.udpports.tcpport2.loglevel=1

If you want to add the new host as a managed device, make sure to use the current total number of managed devices + 1, for the new device number.

To get the current number of managed devices:

 # config -g config.devices.total

Assuming we already have one managed device, our new device will be device 2. Issue the following commands:

 # config -s config. devices.device2.connections.connection1.name=192.168.3.10
 # config -s config. devices.device2.connections.connection1.type=Host
 # config -s config. devices.device2.name=OfficePC
 # config -s config. devices.device2.description=MyPC
 # config -s config.devices.total=2

The following command will synchronize the live system with the new configuration:

 # config -hosts

1.1.6 Trusted Networks

You can further restrict remote access to serial ports based on the source IP address. To configure this via the command line you need to do the following:

Determine the total number of existing trusted network rules (if you have no existing rules) you can assume this is 0

 # config -g config.portaccess.total

This command should display *config.portaccess.total 1*

Note that if you *see config.portaccess.total* this means you have 0 rules configured.

Your new rule will be the existing total plus 1. So if the previous command gave you 0 then you start with rule number 1. If you already have 1 rule your new rule will be number 2 etc.

If you want to restrict access to serial port 5 to computers from a single class C network (*192.168.5.0* say) you need to issue the following commands (assuming you have a previous rule in place).

Add a trusted network:

 # config -s config.portaccess.rule2.address=192.168.5.0
 # config -s "config.portaccess.rule2.description=foo bar"
 # config -s config.portaccess.rule2.netmask=255.255.255.0
 # config -s config.portaccess.rule2.port5=on
 # config -s config.portaccess.total=2

The following command will synchronize the live system with the new configuration:

 # config -r serialconfig

1.1.7 Cascaded Ports

To add a new secondary device with the following settings:

IP address/DNS name	192.168.0.153
Description	CM in office 42
Label	cm7116-5
Number of ports	16

The following commands must be issued:

 # config -s config.cascade.nodes.node1.address=192.168.0.153
 # config -s "config.cascade.nodes.node1.description=CM in office 42"
 # config -s config.cascade.nodes.node1.label=cm7116-5
 # config -s config.cascade.nodes.node1.ports=16

The total number of secondaries must also be incremented. If this is the first secondary being added, type:

 # config -s config.cascade.nodes.total=1

Increment this value when adding more nodes.

NOTE: If a secondary is added using the CLI, then the primary SSH public key will need to be manually copied to every secondary device before cascaded ports will work.

The following command will synchronize the live system with the new configuration:

 # config -r cascade

1.1.8 UPS Connections

Managed UPSes

Before adding a managed UPS, make sure that at least 1 port has been configured to run in 'device mode', and that the device is set to 'ups'.

To add a managed UPS with the following values:

Connected via	Port 1
UPS name	My UPS
Description	UPS in room 5
Username to connect to UPS	User2
Password to connect to UPS	secret
shutdown order	2 (0 shuts down first)
Driver	genericups
Driver option - option	option
Driver option - argument	argument
Logging	Enabled
Log interval	2 minutes
Run script when power is critical	Enabled

 # config -s config.ups.monitors.monitor1.port=/dev/port01
If the port number is higher than 9, eg port 13, enter:
 # config -s config.ups.monitors.monitor1.port=/dev/port13

 # config -s "config.ups.monitors.monitor1.name=My UPS"
 # config -s "config.ups.monitors.monitor1.description=UPS in room 5"
 # config -s config.ups.monitors.monitor1.username=User2

 # config -s config.ups.monitors.monitor1.password=secret
 # config -s config.ups.monitors.monitor1.sdorder=2
 # config -s config.ups.monitors.monitor1.driver=genericups
 # config -s config.ups.monitors.monitor1.options.option1.opt=option
 # config -s config.ups.monitors.monitor1.options.option1.arg=argument
 # config -s config.ups.monitors.monitor1.options.total=1
 # config -s config.ups.monitors.monitor1.log.enabled=on
 # config -s config.ups.monitors.monitor1.log.interval=2
 # config -s config.ups.monitors.monitor1.script.enabled=on

Make sure to increment the total monitors:

 # config -s config.ups.monitors.total=1

The 5 commands below will add the UPS to 'Managed devices. Assuming there are already 2 managed devices configured:

 # config -s "config.devices.device3.connections.connection1.name=My UPS"
 # config -s "config.devices.device3.connections.connection1.type=UPS Unit"
 # config -s "config.devices.device3.name=My UPS"
 # config -s "config.devices.device3.description=UPS in toom 5"
 # config -s config.devices.total=3

To delete this managed UPS:

 # config -d config.ups.monitors.monitor1

Decrement *monitors.total* when deleting a managed UPS

Remote UPSes

To add a remote UPS with the following details (assuming this is our first remote UPS):

UPS name	oldUPS
Description	UPS in room 2
Address	192.168.50.50
Log status	Disabled
Log rate	240 seconds
Run shutdown script	Enabled

 # config -s config.ups.remotes.remote1.name=oldUPS
 # config -s "config.ups.remotes.remote1.description=UPS in room 2"
 # config -s config.ups.remotes.remote1.address=192.168.50.50
 # config -d config.ups.remotes.remote1.log.enabled
 # config -s config.ups.remotes.remote1.log.interval=240
 # config -s config.ups.remotes.remote1.script.enabled=on
 # config -s config.ups.remotes.total=1

The following command will synchronize the live system with the new configuration:

 # config -a

1.1.9 RPC Connections

You can add an RPC connection from the command line but it is not recommended that you do so because of dependency issues.

However before adding an RPC the Management Console GUI code makes sure that at least 1 port has been configured to run in 'device mode', and that the device is set to 'rpc'.

To add an RPC with the following values:

RPC type	APC 7900
Connected via	Port 2
UPS name	MyRPC
Description	RPC in room 5
Login name for device	rpclogin
Login password for device	secret
SNMP community	v1 or v2c
Logging	Enabled
Log interval	600 second
Number of power outlets	4 (depends on the type/model of the RPC)

 # config -s config.ports.port2.power.type=APC 7900
 # config -s config.ports.port2.power.name=MyRPC
 # config -s "config.ports.port2.power.description=RPC in room 5"
 # config -s config.ports.port2.power.username=rpclogin
 # config -s config.ports.port2.power.password=secret
 # config -s config.ports.port2.power.snmp.community=v1
 # config -s config.ports.port2.power.log.enabled=on
 # config -s config.ports.port2.power.log.interval=600
 # config -s config.ports.port2.power.outlets=4

The following five commands are used by the Management Console to add the RPC to 'Managed Devices':

 # config -s config.devices.device3.connections.connection1.name=myRPC
 # config -s "config.devices.device3.connections.connection1.type=RPC Unit"
 # config -s config.devices.device3.name=myRPC
 # config -s "config.devices.device3.description=RPC in room 5"
 # config -s config.devices.total=3

The following command will synchronize the live system with the new configuration:

 # config -a

1.1.10 Environmental

To configure an environmental monitor with the following details:

Monitor name	Envi4
Monitor Description	Monitor in room 5
Temperature offset	2
Humidity offset	5
Enable alarm 1 ?	yes
Alarm 1 label	door alarm
Enable alarm 2 ?	yes
Alarm 2 label	window alarm
Logging enabled ?	yes
Log interval	120 seconds

 # config -s config.ports.port3.enviro.name=Envi4
 # config -s "config.ports.port3.enviro.description=Monitor in room 5"
 # config -s config.ports.port3.enviro.offsets.temp=2
 # config -s config.ports.port3.enviro.offsets.humid=5
 # config -s config.ports.port3.enviro.alarms.alarm1.alarmstate=on
 # config -s config.ports.port3.enviro.alarms.alarm1.label=door alarm

 # config -s config.ports.port3.enviro.alarms.alarm2.alarmstate=on
 # config -s config.ports.port3.enviro.alarms.alarm2.label=window alarm
 # config -s config.ports.port3.enviro.alarms.total=2
 # config -s config.ports.port3.enviro.log.enabled=on
 # config -s config.ports.port3.enviro.log.interval=120

It is important to assign alarms.total=2 even if they are off.

The following 5 commands will add the environmental monitor to 'Managed devices':

To get the total number of managed devices:

 # config -g config.devices.total

Make sure to use the total + 1 for the new device below:

 # config -s config. devices.device5.connections.connection1.name=Envi4
 # config -s "config. devices.device5.connections.connection1.type=EMD Unit"
 # config -s config. devices.device5.name=Envi4
 # config -s "config. devices.device5.description=Monitor in room 5"
 # config -s config.devices.total=5

The following command will synchronize the live system with the new configuration:

 # config -a

1.1.11 Managed Devices

To add a managed device: (also see UPS, RPC connections and Environmental)

 # config -s "config.devices.device8.name=my device"
 # config -s "config.devices.device8.description=The eighth device"
 # config -s "config.devices.device8.connections.connection1.name=my device"
 # config -s config.devices.device8.connections.connection1.type=[serial | Host | UPS | RPC]
 # config -s config.devices.total=8 (decrement this value when deleting a managed device)

To delete the above managed device:

 # config -d config.devices.device8

The following command will synchronize the live system with the new configuration:

 # config -a

1.1.12 Port Log

To configure serial/network port logging:

 # config -s config.eventlog.server.address='remote server ip address'
 # config -s config.eventlog.server.logfacility='facility'

'facility' can be:

Daemon
Local 0-7
Authentication
Kernel
User
Syslog
Mail
News
UUCP

config -s config.eventlog.server.logpriority='priority'

'*priority*' can be:

Info
Alert
Critical
Debug
Emergency
Error
Notice
Warning

Assume the remote log server needs a username 'name1' and password 'secret':

 # config -s config.eventlog.server.username=name1
 # config -s config.eventlog.server.password=secret

To set the remote path as '/opengear/logs' to save logged data:

 # config -s config.eventlog.server.path=/opengear/logs
 # config -s config.eventlog.server.type=[none | syslog | nfs | cifs | usb]

If the server type is set to usb, none of the other values need to be set. The mount point for storing on a remote USB device is */var/run/portmanager/logdir*

The following command will synchronize the live system with the new configuration:

 # config -a

1.1.13 Port Log Password Obfuscation

A config option allows the obfuscation of the response to a user password request during serial port sessions at log levels 2 or 4. This option blots out a password reply (the TX log response) in the /var/run/portX.log logs by substituting the literal password with the string "********". For example:

 # cat /var/log/port01.log
RXDATA: password
*TXDATA: ********

This is managed with a descriptor for ports called password_regex:
config.ports.portX.password_regex

X is the port number against which the regular expression will operated during serial communications.

The regex specified by this descriptor will be used to match the incoming RXDATA line.
If there is a match from the regex, the following TXDATA received by the logger will be obfuscated in the logs.

NOTE: The regex expression must be in Perl compatible syntax (PCRE).

There are three modes of operation:

- Disable all obfuscation on portX:
  config -d config.ports.portX.password_regex

- Match against default regex ("(?i)\bpassword\b") on portX:
  config -s config.ports.portX.password_regex=default

- Match against a custom regex on portX, eg:
  config -s config.ports.portX.password_regex='\bsecret:?\b'

The settings will take effect after the portmanager receives a SIGHUP to force a reload of its configuration, eg with:
pkill -HUP portmanager or a system reboot

1.1.14 Alerts

You can add an email, SNMP or NAGIOS alert by following the steps below.

The general settings for all alerts

Assume this is our second alert, and we want to send alert emails to john@opengear.com and sms's to peter@opengear.com:

 # config -s config.alerts.alert2.description=MySecondAlert
 # config -s config.alerts.alert2.email=john@opengear.com
 # config -s config.alerts.alert2.email2=peter@opengear.com

To use NAGIOS to notify of this alert

 # config -s config.alerts.alert2.nsca.enabled=on

To use SNMP to notify of this alert

 # config -s config.alerts.alert2.snmp.enabled=on

Increment the total alerts:

config -s config.alerts.total=2

Below are the specific settings depending on the type of alert required:

Connection Alert

To trigger an alert when a user connects to serial port 5 or network host 3:

 # config -s config.alerts.alert2.host3='host name'
 # config -s config.alerts.alert2.port5=on
 # config -s config.alerts.alert2.sensor=temp
 # config -s config.alerts.alert2.signal=DSR
 # config -s config.alerts.alert2.type=login

Signal Alert

To trigger an alert when a signal changes state on port 1:

 # config -s config.alerts.alert2.port1=on
 # config -s config.alerts.alert2.sensor=temp
 # config -s config.alerts.alert2.signal=[DSR | DCD | CTS]
 # config -s config.alerts.alert2.type=signal

Pattern Match Alert

To trigger an alert if the regular expression '.*0.0% id' is found in serial port 10's character stream.

*# config -s "config.alerts.alert2.pattern=.*0.0% id"*
 # config -s config.alerts.alert2.port10=on
 # config -s config.alerts.alert2.sensor=temp
 # config -s config.alerts.alert2.signal=DSR
 # config -s config.alerts.alert2.type=pattern

UPS Power Status Alert

To trigger an alert when *myUPS* (on localhost) or *thatUPS* (on remote host *192.168.0.50*) power status changes between on line, on battery and low battery.

 # config -s config.alerts.alert2.sensor=temp
 # config -s config.alerts.alert2.signal=DSR
 # config -s config.alerts.alert2.type=ups
 # config -s config.alerts.alert2.ups1=myUPS@localhost
 # config -s config.alerts.alert2.ups2=thatUPS@192.168.0.50

Environmental and Power Sensor Alert

 # config -s config.alerts.alert2.enviro.high.critical='critical value'
 # config -s config.alerts.alert2.enviro.high.warning='warning value'
 # config -s config.alerts.alert2.enviro.hysteresis='value'
 # config -s config.alerts.alert2.enviro.low.critical='critical value'
 # config -s config.alerts.alert2.enviro.low.warning='warning value'
 # config -s config.alerts.alert2.enviro1='Enviro sensor name'
 # config -s config.alerts.alert2.outlet#='RPCname'.outlet#
'alert2.outlet#' increments sequentially with each added outlet. The second 'outlet#' refers to the specific RPC power outlets.
 # config -s config.alerts.alert2.rpc#='RPC name'
 # config -s config.alerts.alert2.sensor=[temp | humid | load | charge]
 # config -s config.alerts.alert2.signal=DSR
 # config -s config.alerts.alert2.type=enviro
 # config -s config.alerts.alert2.ups1='UPSname@hostname'

Example1: To configure a temperature sensor alert for a sensor called 'SensorInRoom42':

 # config -s config.alerts.alert2.sensor=temp
 # config -s config.alerts.alert2.enviro.high.critical=60
 # config -s config.alerts.alert2.enviro.high.warning=50
 # config -s config.alerts.alert2.enviro.hysteresis=2
 # config -s config.alerts.alert2.enviro.low.critical=5
 # config -s config.alerts.alert2.enviro.low.warning=10
 # config -s config.alerts.alert2.enviro1=SensorInRoom42
 # config -s config.alerts.alert2.signal=DSR
 # config -s config.alerts.alert2.type=enviro

Example2: To configure a load sensor alert for outlets 2 and 4 for an RPC called 'RPCInRoom20':

 # config -s config.alerts.alert2.outlet1='RPCname'.outlet2
 # config -s config.alerts.alert2.outlet2='RPCname'.outlet4
 # config -s config.alerts.alert2.enviro.high.critical=300
 # config -s config.alerts.alert2.enviro.high.warning=280
 # config -s config.alerts.alert2.enviro.hysteresis=20
 # config -s config.alerts.alert2.enviro.low.critical=50
 # config -s config.alerts.alert2.enviro.low.warning=70
 # config -s config.alerts.alert2.rpc1=RPCInRoom20

 # config -s config.alerts.alert2.sensor=load
 # config -s config.alerts.alert2.signal=DSR
 # config -s config.alerts.alert2.type=enviro

Alarm Sensor Alert

To set an alert for 'doorAlarm' and 'windowAlarm' which are two alarms connected to an environmental sensor called 'SensorInRoom3'. Both alarms are disabled on Mondays from 8:15am to 2:30pm:

 # config -s config.alerts.alert2.alarm1=SensorInRoom3.alarm1 (doorAlarm)
 # config -s config.alerts.alert2.alarm1=SensorInRoom3.alarm2 (windowAlarm)
 # config -s config.alerts.alert2.alarmrange.mon.from.hour=8
 # config -s config.alerts.alert2.alarmrange.mon.from.min=15
 # config -s config.alerts.alert2.alarmrange.mon.until.hour=14
 # config -s config.alerts.alert2.alarmrange.mon.until.min=30
 # config -s config.alerts.alert2.description='description'
 # config -s config.alerts.alert2.sensor=temp
 # config -s config.alerts.alert2.signal=DSR
 # config -s config.alerts.alert2.type=alarm

To enable an alarm for the entire day:

 # config -s config.alerts.alert2.alarmrange.mon.from.hour=0
 # config -s config.alerts.alert2.alarmrange.mon.from.min=0
 # config -s config.alerts.alert2.alarmrange.mon.until.hour=0
 # config -s config.alerts.alert2.alarmrange.mon.until.min=0

The following command will synchronize the live system with the new configuration:

 # config -r alerts

1.1.15 SMTP & SMS

To set-up an SMTP mail or SMS server with the following details:

Outgoing server address	mail.opengear.com
Secure connection type	SSL
Sender	John@opengear.com
Server username	john
Server password	secret
Subject line	SMTP alerts

 # config -s config.system.smtp.server=mail.opengear.com
 # config -s config.system.smtp.encryption=SSL (can also be TLS or None)
 # config -s config.system.smtp.sender=John@opengear.com
 # config -s config.system.smtp.username=john
 # config -s config.system.smtp.password=secret
 # config -s config.system.smtp.subject=SMTP alerts

To set-up an SMTP SMS server with the same details as above:

 # config -s config.system.smtp.server2=mail.opengear.com
 # config -s config.system.smtp.encryption2=SSL (can also be TLS or None)
 # config -s config.system.smtp.sender2=John@opengear.com
 # config -s config.system.smtp.username2=john
 # config -s config.system.smtp.password2=secret
 # config -s config.system.smtp.subject2=SMTP alerts

The following command will synchronize the live system with the new configuration:

```
# config -a
```

1.1.16 SNMP

To set-up the SNMP agent on the device:

```
# config -s config.system.snmp.protocol=[ UDP | TCP ]
# config -s config.system.snmp.trapport='port number' (default is 162)
# config -s config.system.snmp.address='NMS IP network address'
# config -s config.system.snmp.commnity='community name' (v1 and v2c only)
# config -s config.system.snmp.engineid='ID' (v3 only)
# config -s config.system.snmp.username='username' (v3 only)
# config -s config.system.snmp.password='password' (v3 only)
# config -s config.system.snmp.version=[ 1 | 2c | 3 ]
```

The following command will synchronize the live system with the new configuration:

```
# config -a
```

1.1.17 Administration

To change the administration settings to:

System Name	og.mydomain.com
System Password (root account)	secret
Description	Device in office 2

```
# config -s config.system.name=og.mydomain.com
# config -P config.users.user1.password          (will prompt user for a password)
# config -s config.system.location="Device in office 2"
```

NOTE: The -P parameter will prompt the user for a password, and encrypt it. In fact, the value of any config element can be encrypted using the -P parameter, but only encrypted user passwords and system passwords are supported. If any other element value were to be encrypted, the value will become inaccessible and will have to be re-set.

An alternative to the second command above is:

```
# /etc/scripts/user-mod -P root
```

The following command will synchronize the live system with the new configuration:

```
# config –a
```

1.1.18 IP settings

To configure the primary network interface with static settings:

IP address	192.168.0.23
Netmask	255.255.255.0
Default gateway	192.168.0.1
DNS server 1	192.168.0.1
DNS server 2	192.168.0.2

```
# config -s config.interfaces.wan.address=192.168.0.23
# config -s config.interfaces.wan.netmask=255.255.255.0
# config -s config.interfaces.wan.gateway=192.168.0.1
# config -s config.interfaces.wan.dns1=192.168.0.1
```

 # config -s config.interfaces.wan.dns2=192.168.0.2
 # config -s config.interfaces.wan.mode=static
 # config -s config.interfaces.wan.media=[Auto | 100baseTx-FD | 100baseTx-HD | 10baseT-HD] 10baseT-FD

To enable bridging between all interfaces:

 # config -s config.system.bridge.enabled=on

The DNS Resolver IPv4/IPv6 priority is set by default to IPv6. You may, if required, edit the DNS resolver precedence configuration file. From the unit command line:

vi /etc/config/resolvpref.conf

uncomment the following line (only if required):

#ipv4_first=true

To enable IPv6 for all interfaces

 # config -s config.system.ipv6.enabled=on

To configure the management lan interface, use the same commands as above but replace:

config.interfaces.wan, with config.interfaces.lan

To enable the management lan interface run the following command:

config -d config.interfaces.lan.disabled
config -r ipconfig

Note: Not all devices have a management LAN interface.

To configure a failover device in case of an outage:

 # config -s config.interfaces.wan.failover.address1='ip address'
 # config -s config.interfaces.wan.failover.address2='ip address'
 # config -s config.interfaces.wan.failover.interface=[eth1 | console | modem]

The network interfaces can also be configured automatically:

 # config -s config.interfaces.wan.mode=dhcp
 # config -s config.interfaces.lan.mode=dhcp

The following command will synchronize the live system with the new configuration:

 # /bin/config –-run=ipconfig

The following command will synchronize the live system with the new configuration:

 # config -r ipconfig

CLI configure for EAPoL

Basic setup with no certificate management required:

 # config -s config.interfaces.lan.eapol.enabled=on
 # config -s config.interfaces.lan.eapol.identity=identity
 # config -s config.interfaces.lan.eapol.method=md5
*# config -s config.interfaces.lan.eapol.password=**********
 # config -r ipconfig

1.1.19 Date & Time settings

To enable NTP using a server at pool.ntp.org issue the following commands:

 # config -s config.ntp.enabled=on

 # config -s config.ntp.server=pool.ntp.org

Alternatively, you can manually change the clock settings:

To change running system time:

 # date 092216452005.05 Format is MMDDhhmm[[CC]YY][.ss]

Then the following command will save this new system time to the hardware clock:

/bin/hwclock --systohc

Alternatively, to change the hardware clock:

 # /bin/hwclock --set --date=092216452005.05 Format is MMDDhhmm[[CC]YY][.ss]

Then the following command will save this new hardware clock time as the system time:

 # /bin/hwclock --hctosys

To change the timezone:

 # config -s config.system.timezone=US/Eastern

The following command will synchronize the live system with the new configuration:

 # config -r time

1.1.20 Dial-in settings

To enable dial-in access on the DB9 serial port from the command line with the following attributes:

Local IP Address	172.24.1.1
Remote IP Address	172.24.1.2
Authentication Type:	MSCHAPv2
Serial Port Baud Rate:	115200
Serial Port Flow Control:	Hardware
Custom Modem Initialization:	ATQ0V1H0
Callback phone	0800223665
User to dial as	user1
Password for user	secret

Run the following commands:

 # config -s config.console.ppp.localip=172.24.1.1
 # config -s config.console.ppp.remoteip=172.24.1.2
 # config -s config.console.ppp.auth=MSCHAPv2
 # config -s config.console.speed=115200
 # config -s config.console.flow=Hardware
 # config -s config.console.initstring=ATQ0V1H0
 # config -s config.console.ppp.enabled=on
 # config -s config.console.ppp.callback.enabled=on
 # config -s config.console.ppp.callback.phone1=0800223665
 # config -s config.console.ppp.username=user1
 # config -s config.console.ppp.password=secret

To make the dialed connection the default route:

 # config -s config.console.ppp.defaultroute=on

Please note that supported authentication types are 'None', 'PAP', 'CHAP' and 'MSCHAPv2'.
Supported serial port baud-rates are '9600', '19200', '38400', '57600', '115200', and '230400'.
Supported parity values are 'None', 'Odd', 'Even', 'Mark' and 'Space'.
Supported data-bits values are '8', '7', '6' and '5'.
Supported stop-bits values are '1', '1.5' and '2'.
Supported flow-control values are 'Hardware', 'Software' and 'None'.

If you do not wish to use out-of-band dial-in access please note that the procedure for enabling start-up messages on the console port is covered in Chapter 2 - Accessing the Console Port.

The following command will synchronize the live system with the new configuration:

 # config –a

1.1.21 DHCP server

To enable the DHCP server on the console management LAN, with settings:

Default lease time	200000 seconds
Maximum lease time	300000 seconds
DNS server1	192.168.2.3
DNS server2	192.168.2.4
Domain name	company.com
Default gateway	192.168.0.1
IP pool 1 start address	192.168.0.20
IP pool 1 end address	192.168.0.100
Reserved IP address	192.168.0.50
MAC to reserve IP for	00:1e:67:82:72:d9
Name to identify this host	John-PC

Issue the commands:

 # config -s config.interfaces.lan.dhcpd.enabled=on
 # config -s config.interfaces.lan.dhcpd.defaultlease=200000
 # config -s config.interfaces.lan.dhcpd.maxlease=300000
 # config -s config.interfaces.lan.dhcpd.dns1=192.168.2.3
 # config -s config.interfaces.lan.dhcpd.dns2=192.168.2.4
 # config -s config.interfaces.lan.dhcpd.domain=company.com
 # config -s config.interfaces.lan.dhcpd.gateway=192.168.0.1
 # config -s config.interfaces.lan.dhcpd.pools.pool1.start=192.168.0.20
 # config -s config.interfaces.lan.dhcpd.pools.pool1.end=192.168.0.100
 # config -s config.interfaces.lan.dhcpd.pools.total=1
 # config -s config.interfaces.lan.dhcpd.staticips.staticip1.ip=192.168.0.50
 # config -s config.interfaces.lan.dhcpd.staticips.staticip1.mac=00:1e:67:82:72:d9
 # config -s config.interfaces.lan.dhcpd.staticips.staticip1.host=John-PC
 # config -s config.interfaces.lan.dhcpd.staticips.total=1

The following command will synchronize the live system with the new configuration:

 # config –a

1.1.22 Services

You can manually enable or disable network servers from the command line. For example if you wanted to guarantee the following server configuration:

HTTP Server	Enabled
HTTPS Server	Disabled
Telnet Server	Disabled
SSH Server	Enabled
SNMP Server	Disabled
Ping Replies (Respond to ICMP echo requests)	Disabled
TFTP server	Enabled

 # config -s config.services.http.enabled=on
 # config -d config.services.https.enabled
 # config -d config.services.telnet.enabled
 # config -s config.services.ssh.enabled=on
 # config -d config.services.snmp.enabled
 # config -d config.services.pingreply.enabled
 # config -s config.services.tftp.enabled=on

To set secondary port ranges for any service

 # config -s config.services.telnet.portbase='port base number' Default: 2000

 # config -s config.services.ssh.portbase='port base number' *Default: 3000*
 # config -s config.services.tcp.portbase='port base number' *Default: 4000*
 # config -s config.services.rfc2217.portbase='port base number' *Default: 5000*
 # config -s config.services.unauthtel.portbase='port base number *Default: 6000*

The following command will synchronize the live system with the new configuration:

config -a

Configure SSH MaxStartups

To set and apply **MaxStartups** specify three colon separated values "start:rate:full" (e.g. "10:20:50"):

 # config -s config.services.ssh.maxstartups=10:20:50
 # config -r sshconfig

To reset **MaxStartups** to default (10:30:60):

 # config -d config.services.ssh.maxstartups
 # config -r sshconfig

To confirm change to **/etc/config/sshd_config**

 # grep "^MaxStartups" /etc/config/sshd_config
MaxStartups 2:4:6

1.1.23 NAGIOS

To configure NAGIOS with the following settings:

NAGIOS host name cm	7116 (Name of this system)
NAGIOS host address	192.168.0.1 (IP to find this device at)
NAGIOS server address	192.168.0.10 (upstream NAGIOS server)
Prefer NRPE over NSCA	Disabled (defaults to Disabled)

 # config -s config.system.nagios.enabled=on
 # config -s config.system.nagios.name=cm7116
 # config -s config.system.nagios.address=192.168.0.1
 # config -s config.system.nagios.server.address=192.168.0.10
 # config -s config.system.nagios.nrpe.prefer="

To configure NRPE with following settings:

NRPE port	5600 (port to listen on for nrpe. Defaults to 5666)
NRPE user	user1 (User to run as. Defaults to nrpe)
NRPE group	group1 (Group to run as. Defaults to nobody)
Allow command arguments	Enabled

 # config -s config.system.nagios.nrpe.enabled=on

 # config -s config.system.nagios.nrpe.port=5600
 # config -s config.system.nagios.user=user1
 # config -s config.system.nagios.nrpe.group=group1
 # config -s config.system.nagios.nrpe.cmdargs=on

To configure NSCA with the following settings:

NSCA encryption	BLOWFISH (can be: [None | XOR | DES | TRPLEDES | CAST-256 | BLOWFISH | TWOFISH | RIJNDAEL-256 | SERPENT | GOST]
NSCA password	secret
NSCA check-in interval	5 minutes
NSCA port	5650 (defaults to 5667)
user to run as	User1 (defaults to nsca)
group to run as	Group1 (defaults to nobody)

 # config -s config.system.nagios.nsca.enabled=on
 # config -s config.system.nagios.nsca.encryption=BLOWFISH
 # config -s config.system.nagios.nsca.secret=secret
 # config -s config.system.nagios.nsca.interval=2
 # config -s config.system.nagios.nsca.port=5650
 # config -s config.system.nagios.nsca.user=User1
 # config -s config.system.nagios.nsca.group=Group1

The following command will synchronize the live system with the new configuration:

config –a

SUPPORT REPORT

Login to the Opengear CLI as root, or as an admin user and become root with sudo -s

Run the following command to generate the Support Report file:
/etc/scripts/support_report.sh

Using scp, WinScp or similar, copy the Support Report text file to your computer from its location on the Opengear device:/etc/config/support_report

Reference: https://opengear.zendesk.com/hc/en-us/articles/10968326767003-Technical-Support-Report

ADVANCED CONFIGURATION

Opengear *console servers* run the embedded Linux operating system. So *Administrator* class users can configure the *console server* and monitor and manage attached serial console and host devices from the command line using Linux commands and the *config* utility (see Chapter 1).

The Linux kernel in the *console server* also supports GNU *bash* shell script enabling the *Administrator* to run custom scripts. This chapter presents a number of useful scripts and scripting tools including

- *delete-node* which is a general script for deleting users, groups, hosts, UPS's etc
- *ping-detect* which will run specified commands when a specific host stops responding to ping requests

This chapter also covers how to perform advanced and custom management tasks using Opengear commands, Linux commands, and open source tools embedded in the *console server*:

- *portmanager* serial port management
- raw data access to the ports and modems
- *iptables* modifications and updating IP filtering rules
- retrieving status information using SNMP and modifying SNMP with *net-snmpd*
- public key authenticated SSH communications
- SSL, configuring HTTPS and issuing certificates
- using *pmpower* for *NUT* and *PowerMan* power device management
- using *IPMItools*
- REST API which allows inspection of console servers
- CDK custom development kit
- sms server tools
- disable multicasting

2.1 Custom Scripting

The *console server* supports GNU *bash* shell commands (refer *Appendix A*) enabling the *Administrator* to run custom scripts.

2.1.1 Custom script to run when booting

The */etc/config/rc.local* script runs when the system boots. By default, this script file is empty. Use it to add commands you to run at boot time

For example, if to display *hello world*:

```
#!/bin/sh
echo "Hello World!"
```

If this script has been copied from a Windows machine you may need to run the following command on the script before *bash* can run:

```
# dos2unix /etc/config/rc.local
```

You may find it useful to call a custom script from the */etc/config/rc.local* file that runs whenever the system is booted.

2.1.2 Running custom scripts when alerts are triggered

When an alert is triggered, specific scripts are called. These scripts reside in */etc/scripts/*.

Here are the default scripts that are run for each applicable alert:

- Connection alert (when a user connects or disconnects from a port or network host): */etc/scripts/portmanager-user-alert* (for port connections) or */etc/scripts/sdt-user-alert* (for host connections)
- Signal alert (when a signal on a port changes state): */etc/scripts/portmanager-signal-alert*
- Pattern match alert (when a specific regular expression is found in the serial ports character stream): */etc/scripts/portmanager-pattern-alert*
- UPS status alert (when the UPS power status changes between on line, on battery, and low battery): */etc/scripts/ups-status-alert*
- Environmental, power and alarm sensor alerts(temperature, humidity, power load and battery charge alerts): */etc/scripts/environmental-alert*
- Interface failover alert: */etc/scripts/interface-failover-alert*

Each script checks to see if you have created a custom script to run instead. The code that does this check is shown below (an extract from the file /etc/scripts/portmanager-pattern-alert):

```
# If there's a user-configured script, run it instead
scripts[0]="/etc/config/scripts/pattern-alert.${ALERT_PORTNAME}"
scripts[1]="/etc/config/scripts/portmanager-pattern-alert"
for (( i=0 ; i < ${#scripts[@]} ; i++ )); do
   if [ -f "${scripts[$i]}" ]; then
      exec /bin/sh "${scripts[$i]}"
   fi
done
```

This code shows that there are two alternative scripts that can be run instead of the default one. This code first checks whether a file *"/etc/config/scripts/pattern-alert.${ALERT_PORTNAME}"* exists. The variable ${ALERT_PORTNAME} must be replaced with "port01" or "port13" or whichever port the alert should run for. If this file cannot be found, the script checks whether the file *"/etc/config/scripts/portmanager-pattern-alert"* exists. If either of these files exists the script calls the *exec* command on the first file that it finds and runs that custom file/script instead.

As an example, you can copy the */etc/scripts/portmanager-pattern-alert* script file to */etc/config/scripts/portmanager-pattern-alert*:

```
# cd /
# mkdir /etc/config/scripts (if the directory does not already exist)
# cp /etc/scripts/portmanager-pattern-alert /etc/config/scripts/portmanager-pattern-alert
```

The next step will be to edit the new script file. Firstly, open the file */etc/config/scripts/portmanager-pattern-alert* using vi (or any other editor), and remove the lines that check for a custom script (the code from above) - this will prevent the new custom script from repeatedly calling itself. After these lines have been removed,

edit the file, or add any additional scripting to the file.

2.1.3 Example script - Power cycling on pattern match

For example, if we had an RPC (PDU) connected to port 1 on a *console server* and also have some telecommunications device connected to port 2 and which is powered by the RPC outlet 3. Now assume the telecom device transmits a character stream "EMERGENCY" out on its serial console port every time that it encounters some specific error, and the only way to fix this error is to power cycle the telecom device.

The first step is to setup a pattern-match alert on port 2 to check for the pattern "EMERGENCY".

Next, we need to create a custom script to deal with this alert:

 # cd /
 # mkdir /etc/config/scripts (if the directory does not already exist)
 # cp /etc/scripts/portmanager-pattern-alert /etc/config/scripts/portmanager-pattern-alert

Note: Make sure to remove the *if* statement (which checks for a custom script) from the new script, in order to prevent an infinite loop.

The *pmpower* utility is used to send power commands to RPC device in order to power cycle our telecom device:

 # pmpower -l port01 -o 3 cycle (The RPC is on serial port 1. The telecom device is powered by RPC outlet 3)

We can now append this command to our custom script. This will guarantee that our telecom device will be power cycled every time the console reads the "EMERGENCY" character stream on port 2.

2.1.4 Example script - Multiple email notifications on each alert

If you desire to send more than one email when an alert triggers, you have to create a replacement script using the method described above and add the appropriate lines to your new script.

Currently, there is a script */etc/scripts/alert-email* which gets run from within all the alert scripts (e.g. *portmanager-user-alert or environmental-alert*). The alert-email script is responsible for sending the email. The line which invokes the email script looks as follows:

/bin/sh /etc/scripts/alert-email $suffix &

To send another email to a single address or the same email to many recipients, edit the custom script appropriately. You can follow the examples in any of the seven alert scripts listed above. In particular let's consider the *portmanager-user-alert* script. If you need to send the same alert email to more than one email address, find the lines in the script responsible for invoking the alert-email script, then add the following lines below the existing lines:

export TOADDR="emailaddress@domain.com"
/bin/sh /etc/scripts/alert-email $suffix &

These two lines assign a new email address to TOADDR and invoke the alert-email script in the background.

2.1.5 Deleting configuration values from the CLI

The *delete-node* script is provided to help with deleting nodes from the command line. The *"delete-node"* script takes one argument, the node name you want to delete (e.g. *"config.users.user1"* or *"config.sdt.hosts.host1"*).

So *delete-node* is a general script for deleting any node you desire (users, groups, hosts, UPS's etc) from the command line. The script deletes the specified node and shuffles the remainder of the node values.

For example if we have five users configured and we use the script to delete user 3, then user 4 will become user 3, and user 5 will become user 4.

This creates an obvious complication as this script does NOT check for any other dependencies that the node being deleted may have had. So you are responsible for making sure that any references and dependencies connected to the deleted node are removed or corrected in the config.xml file.

The script treats all nodes the same. The syntax to run the script is *# ./delete-node {node name}* so to remove user 3:

 # ./delete-node config.users.user3

The *delete-node* script

```
#!/bin/bash
#User must provide the node to be removed. e.g. "config.users.user1"
```

```
# Usage: delete-node {full node path}

if [ $# != 1 ]
then
        echo "Wrong number of arguments"
        echo "Usage: delnode {full '.' delimited node path}"
        exit 2
fi

# test for spaces
TEMP=`echo "$1" | sed 's/.* .*/N/'`
if [ "$TEMP" = "N" ]
then
        echo "Wrong input format"
        echo "Usage: delnode {full '.' delimited node path}"
        exit 2
fi

# testing if node exists
TEMP=`config -g config | grep "$1"`
if [ -z "$TEMP" ]
then
        echo "Node $1 not found"
        exit 0
fi

# LASTFIELD is the last field in the node path e.g. "user1"
# ROOTNODE is the upper level of the node e.g. "config.users"
# NUMBER is the integer value extracted from LASTFIELD e.g. "1"
# TOTALNODE is the node name for the total e.g. "config.users.total"
# TOTAL is the value of the total number of items before deleting e.g. "3"
# NEWTOTAL is the modified total i.e. TOTAL-1
# CHECKTOTAL checks if TOTAL is the actual total items in .xml

LASTFIELD=${1##*.}
ROOTNODE=${1%.*}
NUMBER=`echo $LASTFIELD | sed 's/^[a-zA-Z]*//g'`
TOTALNODE=`echo ${1%.*} | sed 's/\(.*\)/\1.total/'`
TOTAL=`config -g $TOTALNODE | sed 's/.* //'`
NEWTOTAL=$[ $TOTAL -1 ]

# Make backup copy of config file
cp /etc/config/config.xml /etc/config/config.bak
echo "backup of /etc/config/config.xml saved in /etc/config/config.bak"

if [ -z $NUMBER ] # test whether a singular node is being \
#deleted e.g. config.sdt.hosts
then
        echo "deleting $1"
```

```
        config -d "$1"

        echo Done
        exit 0

elif [ $NUMBER = $TOTAL ] # Test if only one item exists
then
        echo "only one item exists"
        # Deleting node
        echo "Deleting $1"
        config -d "$1"

        # Modifying item total.
        config -s "$TOTALNODE=0"

        echo Done
        exit 0

elif [ $NUMBER -lt $TOTAL ] # more than one item exists
then

        # Modify the users list so user numbers are sequential
        # by shifting the users into the gap one at a time...

        echo "Deleting $1"

        LASTFIELDTEXT=`echo $LASTFIELD | sed 's/[0-9]//g'`
        CHECKTOTAL=`config -g $ROOTNODE.$LASTFIELDTEXT$TOTAL`

        if [ -z "$CHECKTOTAL" ]
        then
                echo "WARNING: "$TOTALNODE" greater than number of items"
        fi

        COUNTER=1
        while [ $COUNTER != $((TOTAL-NUMBER+1)) ]
        do

                config -g $ROOTNODE.$LASTFIELDTEXT$((NUMBER+COUNTER)) \
                | while read LINE
                do
                        config -s \
                        "`echo "$LINE" | sed -e "s/$LASTFIELDTEXT$((NUMBER+ \
                        COUNTER))/$LASTFIELDTEXT$((NUMBER+COUNTER-1))/" \
                        -e 's / /=/'`"

                done

                let COUNTER++
        done

        # deleting last user
        config -d $ROOTNODE.$LASTFIELDTEXT$TOTAL
```

```
        # Modifying item total.
        config -s "$TOTALNODE=$NEWTOTAL"

        echo Done
        exit 0
else
        echo "error: item being deleted has an index greater than total items. Increase the total count
variable."
        exit 0
fi
```

2.1.6 Power cycle any device upon a ping request failure

The *ping-detect* script is designed to run specified commands when a monitored host stops responding to ping requests.

The first parameter taken by the *ping-detect* script is the hostname / IP address of the device to ping. Any other parameters are then regarded as a command to run whenever the ping to the host fails. *ping-detect* can run any number of commands.

Below is an example using *ping-detect* to power cycle an RPC (PDU) outlet whenever a specific host fails to respond to a ping request. The *ping-detect* is run from */etc/config/rc.local* to make sure that the monitoring starts whenever the system boots.

So if we assume we have a serially controlled RPC connected to port01 on a *console server* and have a router powered by outlet 3 on the RPC (and the router has an internal IP address of 192.168.22.2). The following instructions will show you how to continuously ping the router and when the router fails to respond to a series of pings, the *console server* will send a command to RPC outlet 3 to power cycle the router, and write the current date/time to a file:

- Copy the *ping-detect* script to */etc/config/scripts /* on the *console server*
- Open */etc/config/rc.local* using vi
- Add the following line to *rc.local*:

  /etc/config/scripts/ping-detect 192.168.22.2 /bin/bash -c "pmpower -l port01 -o 3 cycle && date" > /tmp/output.log &

The above command will cause the *ping-detect* script to continuously ping the host at 192.168.22.2 which is the router. If the router crashes it will no longer respond to ping requests. If this happens, the two commands *pmpower* and *date* will run. The output from these commands is sent to the file */tmp/output.log* so that we have some kind of record. The *ping-detect* is also run in the background using the *"&"*.

Remember the *rc.local* script is only run by default when the system boots. You can manually run the *rc.local* script or the *ping-detect* script if desired.

The *ping-detect* script

The above is just one example of using the *ping-detect* script. The idea of the script is to run any number of commands when a specific host stops responding to ping requests. Here are details of the *ping-detect* script itself:

```
        #!/bin/sh
        # Usage: ping-detect HOST [COMMANDS...]
        # This script takes 2 types of arguments: hostname/IPaddress to ping, and the commands to
        # run if the ping fails 5 times in a row. This script can only take one host/IPaddress per
        # instance. Multiple independent commands can be sent to the script. The commands will be
        # run one after the other.
        #
```

```
# PINGREP is the entire reply from the ping command
# LOSS is the percentage loss from the ping command
# $1 must be the hostname/IPaddress of device to ping
# $2... must be the commands to run when the pings fail.
COUNTER=0
TARGET="$1"
shift
# loop indefinitely:
while true
do
        # ping the device 10 times
        PINGREP=`ping -c 10 -i 1 "$TARGET" `
        #get the packet loss percentage
        LOSS=`echo "$PINGREP" | grep "%" | sed -e 's/.* \([0-9]*\)% .*/\1/'`
        if [ "$LOSS" -eq "100" ]
        then
                COUNTER=`expr $COUNTER + 1`
        else
                COUNTER=0
                sleep 30s
        fi

        if [ "$COUNTER" -eq 5 ]
        then
                COUNTER=0
                "$@"
                sleep 2s
        fi
done
```

2.1.7 Running custom scripts when a configurator is invoked

A configurator is responsible for reading the values in */etc/config/config.xml* and making the appropriate changes live. Some changes made by the configurators are part of the Linux configuration itself such as user passwords or *ipconfig*.

Currently there are nineteen configurators each one responsible for a specific group of config e.g. the *"users"* configurator makes the user configurations in the *config.xml file* live. To see all the available configurators type the following from a command line prompt:

```
# config
```

When a change is made using the Management Console web GUI the appropriate configurator is automatically run. This can be problematic as if another user/administrator makes a change using the Management Console the configurator could possibly overwrite any custom CLI/linux configurations you may have set.

The solution is to create a custom script that runs after each configurator has run. So after each configurator runs it will check whether that appropriate custom script exists. You can then add any commands to the custom script and they will be invoked after the configurator runs.

The custom scripts must be in the correct location:

```
/etc/config/scripts/config-post-
```

To create an alerts custom script:

```
# cd /etc/config/scripts
# touch config-post-alerts
# vi config-post-alerts
```

This script could be used to recover a specific backup config or overwrite a config or make copies of config files etc.

2.1.8 Backing-up the configuration and restoring using a local USB stick

The ***/etc/scripts/backup-usb*** script has been written to save and load custom configuration using a USB flash disk. Before saving configuration locally, you must prepare the USB storage device for use. To do this, disconnect all USB storage devices except for the storage device you wish to use.

Usage: */etc/scripts/backup-usb* COMMAND [FILE]

COMMAND:

check-magic -- check volume label
set-magic -- set volume label
save [FILE] -- save configuration to USB
delete [FILE] -- delete a configuration tarbal from USB
list -- list available config backups on USB
load [FILE] -- load a specific config from USB
load-default -- load the default configuration
set-default [FILE] -- set which file becomes the default

The first thing to do is to check if the USB disk has a label:

 # /etc/scripts/backup-usb check-magic

If this command returns "Magic volume not found", then run the following command:

 # /etc/scripts/backup-usb set-magic

To save the configuration:

 # /etc/scripts/backup-usb save config-20May

To check if the backup was saved correctly:

 # /etc/scripts/backup-usb list

If this command does not display *"* config-20May"* then there was an error saving the configuration.

The set-default command takes an input file as an argument and renames it to "default.opg". This default configuration remains stored on the USB disk. The next time you want to load the default config, it will be sourced from the new default.opg file. To set a config file as the default:

 # /etc/scripts/backup-usb set-default config-20May

To load this default:

 # /etc/scripts/backup-usb load-default

To load any other config file:

 # /etc/scripts/backup-usb load {filename}

The */etc/scripts/backup-usb* script can be executed directly with various *COMMANDS* or called from other custom scripts you may create. However it is recommended that you do not customize the */etc/scripts/backup-usb* script itself at all.

2.1.9 Backing-up the configuration off-box

If you do not have a USB on your *console server* you can back up the configuration to an off-box file. Before backing up you need to arrange a way to transfer the backup off-box. This could be via an NFS

share, a Samba (Windows) share to USB storage or copied off-box via the network. If backing up directly to off-box storage, make sure it is mounted.

/tmp is not a good location for the backup except as a temporary location before transferring it off-box. The */tmp* directory will not survive a reboot. The */etc/config* directory is not a good place either, as it will not survive a restore.

Backup and restore should be done by the root user to ensure correct file permissions are set. The config command is used to create a backup tarball:

config -e <Output File>

The tarball will be saved to the indicated location. It will contain the contents of the */etc/config /* directory in an uncompressed and unencrypted form.

Example nfs storage:

 # mount -t nfs 192.168.0.2:/backups /mnt # config -e /mnt/cm7116.config

 # umount/mnt/

Example transfer off-box via scp:

 # config -e /tmp/cm7116.config
 # scp /tmp/cm7116.config username@192.168.0.2:/backups

The config command is also used to restore a backup:

config -i <Input File>

This will extract the contents of the previously created backup to */tmp*, and then synchronize the */etc/config* directory with the copy in */tmp*.

One problem that can crop up here is that there is not enough room in */tmp* to extract files to. The following command will temporarily increase the size of */tmp*:

mount -t tmpfs -o remount,size=2048k tmpfs /var

If restoring to either a new unit or one that has been factory defaulted, it is important to make sure that the process generating SSH keys is either stopped or completed before restoring configuration. If this is not done, then a mix of old and new keys may be put in place.

As SSH uses these keys to avoid man-in-the-middle attacks, logging in may be disrupted.

2.2 Advanced Portmanager

Opengear's *portmanger* program manages the *console server* serial ports. It routes network connection to serial ports, checks permissions, and monitors and logs all the data flowing to/from the ports.

2.2.1 Portmanager commands

pmshell

The *pmshell* command acts similar to the standard *tip* or *cu* commands, but all serial port access is directed *via* the portmanager.

Example: To connect to port 8 *via* the portmanager:

```
# pmshell -l port08
```

pmshell Commands:

Once connected, the pmshell command supports a subset of the '~' escape commands that tip/cu support. For SSH you must prefix the escape with an additional '~' command (i.e. use the '~~' escape)

Send Break: Typing the character sequence '~b' will generate a BREAK on the serial port (if you're doing this over ssh, you'll need to type "~~b")

History: Typing the character sequence '~h' will generate a history on the serial port.

Quit pmshell: Typing the character sequence '~.' will exit from pmshell.

Set RTS to 1 run the command: *pmshell --rts=1*

Show all signals: *# pmshell –signals*

3DSR=1 DTR=1 CTS=1 RTS=1 DCD=0

Read a line of text from the serial port: *# pmshell –getline*

Note: V3.5.2 and later firmware has includes *pmshell* chooser escape command so you can now hit *~m* from connected serial port to drop back to *pmshell*

Note: For console servers running firmware V3.11.0 and above *pmshell* has a set of key sequences built in to access things like the power menu, return to the serial port selection menu and so on. Extra controls (key sequences) can be added to the built in set of key sequences and can be configured per serial port. You can have all ports behave the same or selectively add control sequences to ports. The controls can be different from port to port for the same function.

For example, you could configure pmshell such that when you are using serial port 2, pressing *Ctrl+p* would take you straight to the power menu for that port.

The *pmshell* control commands are configured only via the command line.

There is a helper script which will configure a control command on a range of serial ports to eliminate the cumbersome task of entering the configuration command for every port. You will still need to use this script once per control function (see below) but there are only six of these.

pmshell control functions and their built in key sequences:

~b - Generate BREAK - send a break to the console

~h - View history - see the traffic logs for the port - must have port logging enabled

~p - Power menu - open the power menu for the port - port must be configured for an RPC

~c – Port Configuration menu – display configuration menu for currently connected port

~u – User sessions disconnect menu - open list of user sessions, select by number to disconnect

~m - Connect to port menu - go back to the serial port selection menu

~. - Exit pmshell - exit pmshell completely

~? - Show help message - shows the help message

Per Port Control Command Config Parameters:

config.ports.portX.ctrlcode.break - Generate BREAK

config.ports.portX.ctrlcode.portlog - View History

config.ports.portX.ctrlcode.power - Power menu

config.ports.portX.ctrlcode.chooser - Connect to port menu

config.ports.portX.ctrlcode.quit - Exit pmshell

config.ports.portX.ctrlcode.help - Show help message

The *pmshell* help message is NOT updated with the extra control command keys that may be configured. As an example, to configure *Ctrl+p* to open the power menu when using serial port 3, enter the following in the console server's command shell:

config -s config.ports.port3.ctrlcode.power=16

killall -HUP portmanager

The first command sets the power menu command to listen for Ctrl+p (where decimal 16 is the character code sent when you press Ctrl+p in the serial port session - see the range of control codes below). The second command (killall -HUP portmanager) tells portmanager to reload the configuration so that the new control code will take affect - rebooting the device would also work.

There is a script to set serial control codes on a range of ports so that bulk port configuration can be performed more easily. For example to set the power menu control code to CTRL-P (keycode 16) on ports 4 to 10 inclusive, enter the following at the command line:

/etc/scripts/set-serial-control-codes 4 10 power 16

This sets the power menu control key to Ctrl+p (see the range of control codes below). NOTE: If you've not configured anything on a particular serial port in the included range, configuration for that port will be skipped.

Control Codes (Ctrl+a=1 ... Ctrl+z=26):

Ctrl+a = 1

Ctrl+b = 2

Ctrl+c = 3

Ctrl+d = 4

Ctrl+e = 5

Ctrl+f = 6

Ctrl+g = 7

Ctrl+h = 8

Ctrl+i = 9

Ctrl+j = 10

Ctrl+k = 11

Ctrl+l = 12

Ctrl+m = 13

Ctrl+n = 14

Ctrl+o = 15

Ctrl+p = 16

Ctrl+q = 17

Ctrl+r = 18

Ctrl+s = 19

Ctrl+t = 20

Ctrl+u = 21

Ctrl+v = 22

Ctrl+w = 23

Ctrl+x = 24

Ctrl+y = 25

Ctrl+z = 26

pmchat

The *pmchat* command acts similar to the standard *chat* command, but all serial port access is directed *via* the portmanager.

Example: To run a chat script *via* the portmanager:

 # pmchat -v -f /etc/config/scripts/port08.chat < /dev/port08

For more information on using *chat* (and *pmchat*) you should consult the UNIX man pages:

http://techpubs.sgi.com/library/tpl/cgibin/getdoc.cgi?coll=linux&db=man&fname=/usr/share/catman/man8/chat.8.html

pmusers

The *pmusers* command is used to query the portmanager for active user sessions.

Example: To detect which users are currently active on which serial ports:

 # pmusers

This command will output nothing if there are no active users currently connected to any ports, otherwise it will respond with a sorted list of usernames per active port:

Port 1:
user1
user2
Port 2:
user1
Port 8:
user2

The above output indicates that a user named "*user1*" is actively connected to ports 1 and 2, while "*user2*" is connected to both ports 1 and 8

Note: With V3.11 firmware and later the *pmusers* command is extended with the *--disconnect* option, which allows an *admin* user or *root* to disconnect console server sessions from the command line. The following connection types can be disconnected:

telnet
SSH
Raw TCP
Unauth'ed Telnet

You cannot disconnect an *RFC2217* session.

If the *--disconnect* option is specified, the *pmusers* command goes into disconnect mode where you can specify the users with *-u*, the ports with *-l* (by *label*) or *-n* (by *name*).

By default the command will prompt the user before actually disconnecting the matching sessions. This can be overriden with the *--no-prompt* argument.

Example: *pmuser* sessions:

 # pmusers --disconnect

Disconnect all users from all ports? (y/n)
y
5 sessions were disconnected

 # pmusers --disconnect -u robertw
Disconnect user robertw from all ports? (y/n)
y
1 session was disconnected

 # pmusers --disconnect -u robertw -n 5
Disconnect user robertw from port 5 (BranchRouter01)? (y/n)
y
No sessions were disconnected

 # pmusers --disconnect -n 5
Disconnect all users from port 5 (BranchRouter01)? (y/n)
y
2 sessions were disconnected

 # pmusers --disconnect -u robertw -u pchunt -n 4 -n 6
Disconnect users robertw, pchunt from ports 4, 6? (y/n)
y
10 sessions were disconnected

 # pmusers --disconnect -u tester --no-prompt
No sessions were disconnected

portmanager daemon

There is normally no need to stop and restart the daemon. To restart the daemon normally, just run the command:

 # portmanager

Supported command line options are:

Force portmanager to run in the foreground: *--nodaemon*

Set the level of debug logging: *--loglevel={debug,info,warn,error,alert}*

Change which configuration file it uses: *-c /etc/config/portmanager.conf*

Signals

Sending a SIGHUP signal to the portmanager will cause it to re-read its configuration file

2.2.2 External Scripts and Alerts

The portmanager has the ability to execute external scripts on certain events.

When a port is opened by the portmanager:

- When the portmanager opens a port, it attempts to execute */etc/config/scripts/portXX.init* (where XX is the number of the port, *e.g.* 08). The script is run with STDIN and STDOUT both connected to the serial port.
- If the script cannot be executed, then portmanager will execute */etc/config/scripts/portXX.chat via* the chat command on the serial port.

When an alert occurs on a port:

- When an alert occurs on a port, the portmanager will attempt to execute */etc/config/scripts/portXX.alert* (where XX is the port number, e.g. 08)
- The script is run with STDIN containing the data which triggered the alert, and STDOUT redirected to /dev/null, NOT to the serial port. If you wish to communicate with the port, use *pmshell* or *pmchat* from within the script.
- If the script cannot be executed, then the alert will be mailed to the address configured in the system administration section.

When a user connects to any port:

- If a file called */etc/config/pmshell-start.sh* exists it is run when a user connects to a port. It is provided 2 arguments, the "Port number" and the "Username". Here is a simple example:

```
</etc/config/pmshell-start.sh >
#!/bin/sh
PORT="$1"
USER="$2"
echo "Welcome to port $PORT $USER"
< /etc/config/pmshell-start.sh>
```

- The return value from the script controls whether the user is accepted or not, if 0 is returned (or nothing is done on exit as in the above script) the user is permitted, otherwise the user is denied access.
- Here is a more complex script which reads from configuration to display the port label if available and denies access to the root user:

```
</etc/config/pmshell-start.sh>
#!/bin/sh
PORT="$1"
USER="$2"
LABEL=$(config -g config.ports.port$PORT.label | cut -f2- -d' ')
if [ "$USER" == "root" ]; then
        echo "Permission denied for Super User"
        exit 1
fi
if [ -z "$LABEL" ]; then
echo "Welcome $USER, you are connected to Port $PORT"
else
echo "Welcome $USER, you are connected to Port $PORT ($LABEL)"
fi
</etc/config/pmshell-start.sh>
```

2.3 Raw Access to Serial Ports

2.3.1 Access to serial ports

You can use *tip* and *stty* to completely bypass the *portmanager* and have raw access to the serial ports.

When you run *tip* on a *portmanager* controlled port, *portmanager* closes that port, and stops monitoring it until *tip* releases control of it.

With *stty*, the changes made to the port only "stick" until that port is closed and opened again. So it is doubtful that people will want to use *stty* for more than initial debugging of the serial connection.

If you want to use *stty* to configure the port, you can put *stty* commands in */etc/config/scripts/portXX.init* which gets run whenever portmanager opens the port.

Otherwise, any setup you do with *stty* will get lost when the portmanager opens the port. (the reason that portmanager sets things back to its *config* rather than using whatever is on the port, is so the port is in a known good state, and will work, no matter what things are done to the serial port outside of portmanager).

2.3.2 Accessing the console/modem port

The console dial-in is handled by *mgetty*, with automatic PPP login extensions. *mgetty* is a smart *getty* replacement, designed to be used with Hayes compatible data and data/fax modems. *mgetty* knows about modem initialization, manual modem answering (so your modem doesn't answer if the machine isn't ready), UUCP locking (so you can use the same device for dial-in and dial-out). *mgetty* provides very extensive logging facilities. All standard *mgetty* options are supported.

Modem initialization strings:

- To override the standard modem initialization string either use the Management Console or the command line config tool.

Enabling Boot Messages on the Console:

- If you are not using a modem on the DB9 console port and instead wish to connect to it directly via a Null Modem cable you may want to enable verbose mode allowing you to see the standard linux start-up messages. This can be achieved with the following commands:

  # /bin/config --set=config.console.debug=on # /bin/config --run=console # reboot

- If at some point in the future you chose to connect a modem for dial-in out-of-band access the procedure can be reversed with the following commands.

  # /bin/config --del=config.console.debug # /bin/config --run=console # reboot

2.4 IP Filtering

The *console server* uses the *iptables* utility to provide a stateful firewall of LAN traffic. By default rules are automatically inserted to allow access to enabled services, and serial port access *via* enabled protocols. The commands which add these rules are contained in configuration files:

/etc/config/fw.rules

This is an executable shell script which is run whenever the LAN interface is brought up and whenever modifications are made to the *iptables* configuration as a result of CGI actions or the *config* command line tool.

The basic steps performed are as follows:

- Running *iptables* configuration is erased, per-interface and other standard system chains are installed
- Fall through *Block* rules (default deny) are installed
- **Serial & Network: Services** policies are installed in per-interface chains
- Custom **Serial & Network: Firewall** rules are inserted at the top of the rule sets, taking priority over any other configuration

If you require further firewall customization, extra rules can be persisted by creating a file at **/etc/config/scripts/firewall-post** containing *iptables* commands to amend the firewall policy.

There's good documentation about using the *iptables* command at the Linux *netfilter* website *http://netfilter.org/documentation/index.html*.There are also many high-quality tutorials and HOWTOs available *via* the *netfilter* website, in particular peruse the tutorials listed on the *netfilter* HOWTO page.

2.5 SNMP Status Reporting

All console servers contain an SNMP Service (*snmpd*) which can provide status information on demand. *snmpd* is an SNMP agent which binds to a port and awaits requests from SNMP management software. Upon receiving a request, it processes the request(s), collects the requested information and/or performs the requested operation(s) and returns the information to the sender.

Note: Initially only advanced *console server* models were equipped with an SNMP Service. With V3.0 (and later) firmware this support was extended to all *console servers*. Also the MIBS were extended (and renamed for compliance) with this firmware release.

All console servers can also be configured to send SNMP traps/messages to multiple remote SNMP Network Managers on defined trigger events.

2.5.1 Retrieving status information using SNMP

Console servers can provide serial and device status information through SNMP. This includes

- Serial port status
- Active users
- Remote Power Control (RPC) and Power Distribution Unit (PDU) status
- Environmental Monitoring Device (EMD) status
- Signal alert status
- Environmental alert status and
- UPS alert status

The MIBs in your *console server* are located in */etc/snmp/mibs.* You also can view the current MIBs online at / and they include:

OG-STATUS-MIB	This MIB contains serial and connected device status information (for *snmpstatusd* & *snmpalertd*)
OG-STATUSv2-MIB	This new MIB contains extended status and alert
OG-SMI-MIB	Enterprise structure of management information
OGTRAP-MIB	SMIv1 traps from old MIBS (as *smilin*t will not let SMIv1 structures coexist with SMIv2)
OGTRAPv2-MIB	Updated traps

2.5.2 Check firewall rules

- Select **System: Services** and ensure the *SNMP daemon* box has been checked for the interface required. This will allow SNMP requests through the firewall for the specified interface.

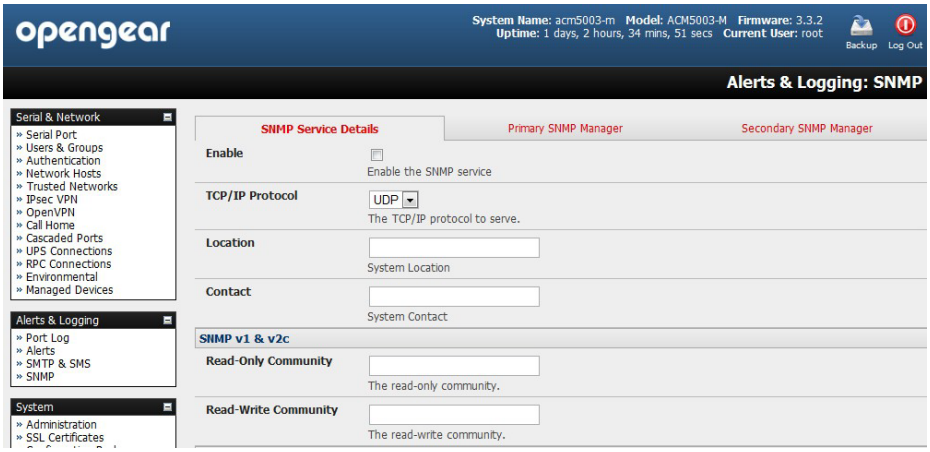
2.5.3 Enable SNMP Service

The *console server* supports different versions of SNMP including SNMPv1, SNMPv2c and SNMPv3.

SNMP, although an industry standard, brings with it a variety of security concerns. For example, SNMPv1 and SNMPv2c offer no inherent privacy, while SNMPv3 is susceptible to man-in-the-middle attacks. Recent IETF developments suggests tunnelling SNMP over widely accepted technologies such as SSH (Secure Shell) or TLS (Transport Layer Security) rather than relying on a less mature security systems such as SNMPv3's USM (User-based Security Model).

Additional information regarding SNMP security issues and SNMPv3 can be found at:
http://net-snmp.sourceforge.net/wiki/index.php/TUT:Security
http://www.ietf.org/html.charters/snmpv3-charter.html.

- Select **Alerts & Logging: SNMP**



- The **SNMP Service Details** tab is shown by default. The SNMP Service Details tab controls aspects of the SNMP Service including Security Level. It manages requests from external agents for Opengear status information.
- Check the **Enable the SNMP Service** box to start the SNMP Service. The Service is disabled by default.
- Select either **UDP** or **TCP** for the TCP/IP Protocol. UDP is the recommended protocol and is selected by default. TCP should only be used in special cases such as when Port Forwarding SNMP requests/responses to or from the Opengear device is required.
- Complete the **Location** and **Contact** fields. The Location field should describe the physical location of the Opengear and will be used in response to requests for the SNMPv2-MIB::sysLocation.0 of the device. The Contact field refers to the person responsible for the Opengear such as the System Administrator and will be used in response to requests as follows: SNMPv2-MIB::sysContact.0.
- Enter the **Read-Only Community** and **Read-Write Community.** This is required for **SNMP v1 & v2c** only. The Read-Only Community field is used to specify the SNMPv1 or SNMPv2c community that will be allowed read-only (GET and GETNEXT) access. This must be specified in order for both versions to become enabled. The Read-Write Community field is used to specify the SNMPv1 or SNMPv2c community that will be allowed read-write (GET, GETNEXT and SET) access.
- Configure **SNMP v3,** if required. SNMP v3 provides secure SNMP operations through the use of USM (User-based Security Model). It offers various levels of security including user-based authentication and basic encryption.
    - The **Engine ID** is used to localize the SNMPv3 user. It will be automatically generated from a Network Interface (eth0) hardware address, if left blank, or must be entered as a hex value e.g. 0x01020304.

- Specify the **Security Level**:

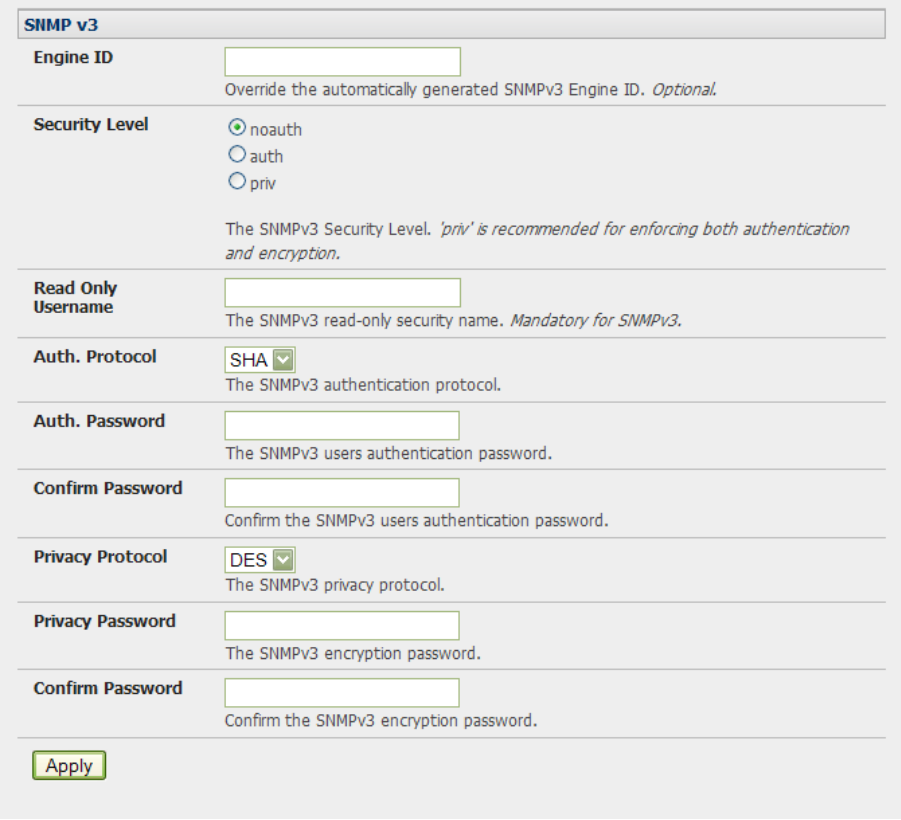
  noauth No authentication or encryption is required. This is the minimum level of security.

  auth Authentication will be required but encryption is not enforced. An authentication protocol (SHA or MD5) and password will be required.

  priv Enforces the use of encryption. This is the highest level of security and requires an encryption protocol (DES or AES) and password in addition to the authentication protocol and password.

- Complete the **Read Only Username.** Enter the read only security name. This field is mandatory and must be completed when configuring the *console server* for SNMPv3.
- For a **Security Level** of **auth**, select the **Auth. Protocol (SHA** or **MD5)** and the **Auth. Password**. A password of at least 8 characters is required.
- For a **Security Level** of **priv**, select the **Privacy Protocol (DES** or **AES)** and the **Privacy Password**. **AES** is recommended as it provides stronger privacy but requires more intense calculations. A password of at least 8 characters is required.

▶ Click **Apply**



▶ Setup serial ports and devices as per operational requirements such as UPS, RPC/PDU and EMD

▶ Copy the mibs from /etc/snmp/mibs on the Opengear product to a local directory using *scp* or W*inscp*. For example:

*scp root@im4004:/etc/snmp/mibs/**

- Using the *snmpwalk* and *snmpget* commands, the status information can be retrieved from any console server. For example:

  snmpwalk -Oa -v1 -M .:/usr/share/snmp/mibs -c public im4004 OG-STATUS-MIB::ogStatus

```
OG-STATUS-MIB::ogSerialPortStatusPort.1 = INTEGER: 2
OG-STATUS-MIB::ogSerialPortStatusPort.2 = INTEGER: 3
OG-STATUS-MIB::ogSerialPortStatusPort.3 = INTEGER: 4
OG-STATUS-MIB::ogSerialPortStatusSpeed.0 = INTEGER: 9600
OG-STATUS-MIB::ogSerialPortStatusSpeed.1 = INTEGER: 9600
OG-STATUS-MIB::ogSerialPortStatusSpeed.2 = INTEGER: 19200
OG-STATUS-MIB::ogSerialPortStatusSpeed.3 = INTEGER: 9600
OG-STATUS-MIB::ogSerialPortStatusDCD.0 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDCD.1 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDCD.2 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDCD.3 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDTR.0 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDTR.1 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDTR.2 = INTEGER: on(1)
OG-STATUS-MIB::ogSerialPortStatusDTR.3 = INTEGER: on(1)
OG-STATUS-MIB::ogSerialPortStatusDSR.0 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDSR.1 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDSR.2 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusDSR.3 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusCTS.0 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusCTS.1 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusCTS.2 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusCTS.3 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusRTS.0 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusRTS.1 = INTEGER: off(0)
OG-STATUS-MIB::ogSerialPortStatusRTS.2 = INTEGER: on(1)
OG-STATUS-MIB::ogSerialPortStatusRTS.3 = INTEGER: on(1)
OG-STATUS-MIB::ogRpcStatusName.0 = STRING: baytech
OG-STATUS-MIB::ogRpcStatusMaxTemp.0 = INTEGER: 0
OG-STATUS-MIB::ogRpcStatusAlertCount.0 = INTEGER: 0
OG-STATUS-MIB::ogEmdStatusName.0 = STRING: EMD_test
OG-STATUS-MIB::ogEmdStatusTemp.0 = INTEGER: 0
OG-STATUS-MIB::ogEmdStatusHumidity.0 = INTEGER: 0
OG-STATUS-MIB::ogEmdStatusAlertCount.0 = INTEGER: 0
OG-STATUS-MIB::ogSignalAlertStatusPort.0 = INTEGER: 4
OG-STATUS-MIB::ogSignalAlertStatusLabel.0 = STRING: port04
OG-STATUS-MIB::ogSignalAlertStatusSignalName.0 = STRING: DSR
OG-STATUS-MIB::ogSignalAlertStatusState.0 = INTEGER: on(1)
OG-STATUS-MIB::ogEnvAlertStatusDevice.0 = STRING: EMD_test
OG-STATUS-MIB::ogEnvAlertStatusDevice.1 = STRING: EMD_test
OG-STATUS-MIB::ogEnvAlertStatusSensor.0 = STRING: a2
OG-STATUS-MIB::ogEnvAlertStatusSensor.1 = STRING: temp
OG-STATUS-MIB::ogEnvAlertStatusOutlet.0 = INTEGER: 0
OG-STATUS-MIB::ogEnvAlertStatusOutlet.1 = INTEGER: 0
OG-STATUS-MIB::ogEnvAlertStatusValue.0 = INTEGER: 1
OG-STATUS-MIB::ogEnvAlertStatusValue.1 = INTEGER: 21
OG-STATUS-MIB::ogEnvAlertStatusOldValue.0 = INTEGER: 0
OG-STATUS-MIB::ogEnvAlertStatusOldValue.1 = INTEGER: 3
OG-STATUS-MIB::ogEnvAlertStatusStatus.0 = INTEGER: 1
OG-STATUS-MIB::ogEnvAlertStatusStatus.1 = INTEGER: 5
```

snmpget -Oa -v1 -M .:/usr/share/snmp/mibs -c public im4004 OG-STATUSMIB:: ogSerialPortStatusSpeed.2

```
OG-STATUS-MIB::ogSerialPortStatusSpeed.2 = INTEGER: 19200
```

noauth

snmpwalk -Oa –v3 –l noAuthNoPriv –u readonlyusername -M .:/usr/share/snmp/mibs im4004 OG-STATUS-MIB::ogStatus

auth

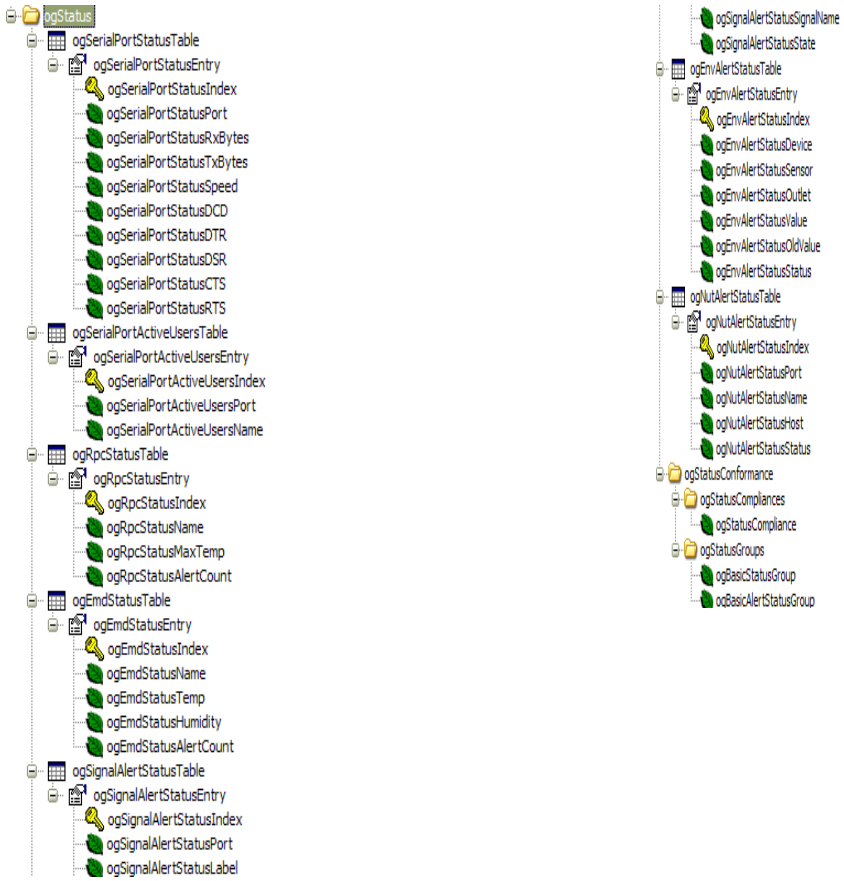
snmpwalk -Oa –v3 –l authNoPriv –u readonlyusername –a SHA –A "authpassword" -M .:/usr/share/snmp/mibs im4004 OG-STATUS-MIB::ogStatus

priv

snmpwalk -Oa –v3 –l authNoPriv –u readonlyusername –a SHA –A "authpassword" –x DES –X "privpassword" -M .:/usr/share/snmp/mibs im4004 OG-STATUS-MIB::ogStatus

-l	Security Level
-u	Security Name or Read Only Username
-a	Authentication Protocol – SHA or MD5
-A	Authentication Password
-x	Privacy Protocol – DES or AES
-X	Privacy Password

A mib browser may be used to explore the Opengear enterprise MIB structure. For example, the *ogStatus* tree is shown below:



2.5.4 Adding multiple remote SNMP managers

You can add multiple SNMP servers for alert traps add the first and second SNMP servers using the Management Console or the command line *config* tool. Further SNMP servers must be added manually using *config.*

Log in to the *console server*'s command line shell as root or an admin user. Refer back to the Management Console UI or user documentation for descriptions of each field.

To set the SNMP Manager Address field:
config –set="config.system.snmp.address3=w.x.y.z"

.. replacing *w.x.y.z* with the IP address or DNS name.

To set the Manager Trap Port field
config --set="config.system.snmp.trapport3=162"
.. replacing 162 with the TCP/UDP port number

To set the SNMP Manager Protocol field:
config --set="config.system.snmp.protocol3=UDP" or
config --set="config.system.snmp.protocol3=TCP"

To set the SNMP Manager Version field:
config --set="config.system.snmp.version3=3"

To set the SNMP Manager v1 & v2c community field:

config --set="config.system.snmp.community3=public"

To set the SNMP Manager v3 Engine ID field:
config –set="config.system.snmp.engineid3=0x8000000001020304"

.. replacing 0x8000000001020304 with the hex Engine-ID

To set the SNMP Manager v3 Security Level field:
config --set="config.system.snmp.seclevel3=noAuthNoPriv" or
config --set="config.system.snmp.seclevel3=authNoPriv" or
config --set="config.system.snmp.seclevel3=authPriv"

To set the SNMP Manager v3 Username field:
config --set="config.system.snmp.username3=username"

To set the SNMP Manager v3 Auth. Protocol and password fields:
config –set="config.system.snmp.authprotocol3=SHA" or

config --set="config.system.snmp.authprotocol3=MD5"
config --set="config.system.snmp.authpassword3=password 1"

To set the SNMP Manager v3 Privacy Protocol and password fields:
config –set="config.system.snmp.privprotocol3=AES" or

config –set="config.system.snmp.privprotocol3=DES"

config --set="config.system.snmp.privpassword3=password 2"

Once the fields are set, apply the configuration with the following command:
config --run snmp

You can add a third or more SNMP servers by incrementing the "2" in the above commands, e.g. config.system.snmp.protocol3, config.system.snmp.address3, etc

2.6 Secure Shell (SSH) Public Key Authentication

This section covers the generation of public and private keys in a Linux and Windows environment and configuring SSH for public key authentication. The steps to use in a Clustering environment are:

- Generate a new public and private key pair
- Upload the keys to the Primary and to each Secondary *console server*
- Fingerprint each connection to validate

2.6.1 SSH Overview

Popular TCP/IP applications such as telnet, rlogin, ftp, and others transmit their passwords unencrypted. Doing this across pubic networks like the Internet can have catastrophic consequences. It leaves the door open for eavesdropping, connection hijacking, and other network-level attacks.

Secure Shell (SSH) is a program to log into another computer over a network, to execute commands in a remote machine, and to move files from one machine to another. It provides strong authentication and secure communications over insecure channels.

OpenSSH is an open source SSH application that encrypts all traffic (including passwords) to effectively eliminate these risks. Additionally, OpenSSH provides a myriad of secure tunneling capabilities, as well as a variety of authentication methods.

The only changes in the Opengear SSH implementation are:

- PAM support

- EGD[1]/PRNGD[2] support and replacements for OpenBSD library functions that are absent from other versions of UNIX
- The config files are now in */etc/config. e.g.*
  - */etc/config/sshd_config* instead of */etc/sshd_config*
  - */etc/config/ssh_config* instead of */etc/ssh_config*
  - */etc/config/users/<username>/.ssh* / instead of */home/<username>/.ssh/*

2.6.2 Generating Public Keys (Linux)

To generate new SSH key pairs use the Linux *ssh-keygen* command. This will produce an RSA or DSA public/private key pair and you will be prompted for a path to store the two key files e.g. *id_dsa.pub* (the public key) and *id_dsa* (the private key). For example:

$ ssh-keygen -t [rsa|dsa]
Generating public/private [rsa|dsa] key pair.
Enter file in which to save the key *(/home/user/.ssh/id_[rsa|dsa]):*
Enter *passphrase* (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in */home/user/.ssh/id_[rsa|dsa].*
Your public key has been saved in */home/user/.ssh/id_[rsa|dsa].pub.*
The key fingerprint is:
28:aa:29:38:ba:40:f4:11:5e:3f:d4:fa:e5:36:14:d6 user@server
$

It is advisable to create a new directory to store your generated keys. It is also possible to name the files after the device they will be used for. For example:

$ mkdir keys
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): /*home/user/keys/control_room*
Enter *passphrase* (empty for no passphrase):
Enter same *passphrase* again:
Your identification has been saved in */home/user/keys/control_room*
Your public key has been saved in */home/user/keys/control_room.pub*.
The key fingerprint is:
28:aa:29:38:ba:40:f4:11:5e:3f:d4:fa:e5:36:14:d6 user@server
$

You must ensure there is no password associated with the keys. If there is a password, then the Opengear devices will have no way to supply it as runtime.

Full documentation for the *ssh-keygen* command can be found at *http://www.openbsd.org/cgi-bin/man.cgi?query=ssh-keygen*

2.6.3 Installing the SSH Public/Private Keys (Clustering)

For Opengear *console servers* the keys can be simply uploaded through the web interface, on the **System: Administration** page. This enables you to upload stored RSA or DSA Public Key pairs to the Primary and apply the Authorized key to the secondary. Once complete you then proceed to Fingerprinting as described below.

2.6.4 Installing SSH Public Key Authentication (Linux)

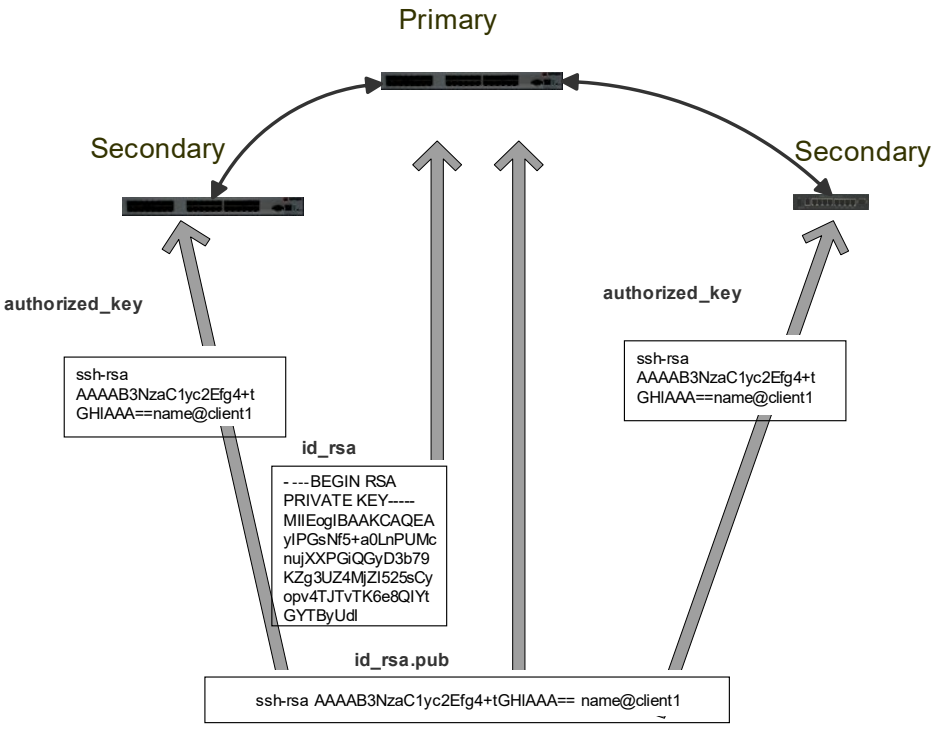
Alternately the public key can be installed on the unit remotely from the linux host with the *scp* utility as follows.

Assuming the user on the Management Console is called "fred"; the IP address of the *console server* is 192.168.0.1 (default); and the public key is on the *linux/unix* computer in *~/.ssh/id_dsa.pub*. Execute the following command on the *linux/unix* computer:

*scp ~/.ssh/id_dsa.pub *
root@192.168.0.1:/etc/config/users/fred/.ssh/authorized_keys

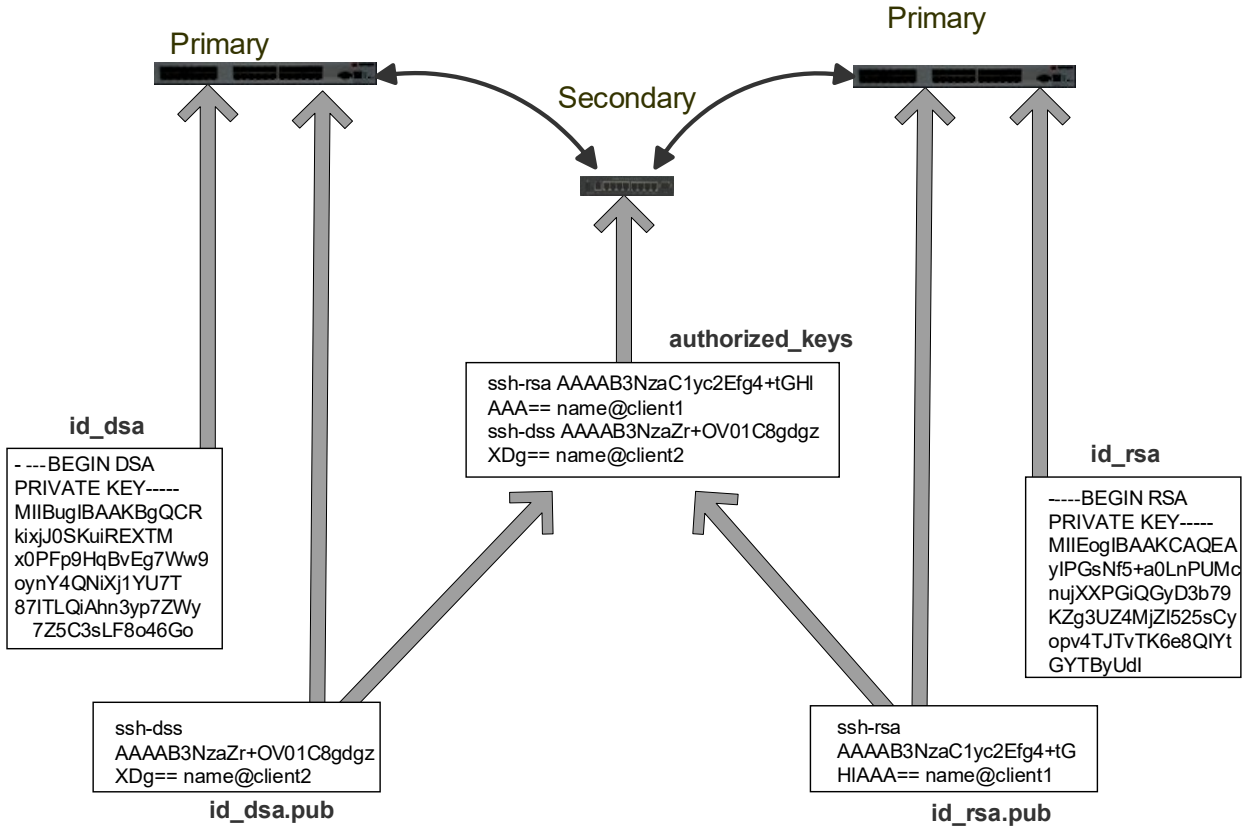
The authorized_keys file on the *console server* needs to be owned by "fred", so login to the Management Console as **root** and type:

chown fred /etc/config/users/fred/.ssh/authorized_keys



If the Opengear device selected to be the server will only have one client device, then the *authorized_keys* file is simply a copy of the public key for that device. If one or more devices will be clients of the server, then the *authorized_keys* file will contain a copy of all of the public keys. RSA and DSA keys may be freely

mixed in the authorized_keys file. For example, assume we already have one server, called bridge_server, and two sets of keys, for the control_room and the plant_entrance:
$ ls /home/user/keys control_room control_room.pub plant_entrance plant_entrance.pub $ cat /home/user/keys/control_room.pub /home/user/keys/plant_entrance.pub > /home/user/keys/authorized_keys_bridge_server



More documentation on OpenSSH can be found at:
http://openssh.org/portable.html
http://www.openbsd.org/cgi-bin/man.cgi?query=ssh&sektion=1
http://www openbsd.org/cgi-bin/man.cgi?query=sshd.

2.6.5 Generating public/private keys for SSH (Windows)

This section describes how to generate and configure SSH keys using Windows.

First create a new user from the Opengear Management (the following example uses a user called "testuser") making sure it is a member of the "users" group.

If you do not already have a public/private key pair you can generate them now using *ssh-keygen*, *PuTTYgen* or a similar tool:

PuTTYgen: *http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html*

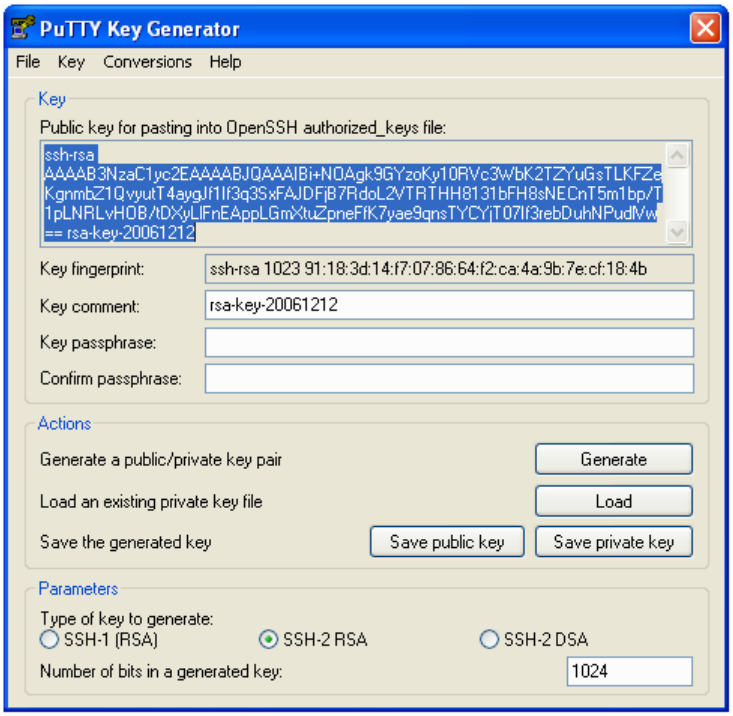
OpenSSH: *http://www.openssh.org/*

OpenSSH (Windows): *http://sshwindows.sourceforge.net/download/*

For example using PuTTYgen, make sure you have a recent version of the *puttygen.exe* (available from *http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)* Make sure you have a recent version of WinSCP (available from *http://winscp.net/eng/download.php*)

To generate a SSH key using PuTTY *http://sourceforge.net/docs/F02/#clients:*

- Execute the PUTTYGEN.EXE program
- Select the desired key type *SSH2 DSA* (you may use RSA or DSA) within the *Parameters* section
- It is important that you leave the passphrase field blank
- Click on the *Generate* button
- Follow the instruction to move the mouse over the blank area of the program in order to create random data used by PUTTYGEN to generate secure keys. Key generation will occur once PUTTYGEN has collected sufficient random data



- Create a new file " *authorized_keys* " (with notepad) and copy your public key data from the "Public key for pasting into OpenSSH authorized_keys file" section of the PuTTY Key Generator, and paste the key data to the "authorized_keys" file. Make sure there is only one line of text in this file
- Use WinSCP to copy this "authorized_keys" file into the user's home directory: eg. */etc/config/users/testuser/.ssh/authorized_keys* of the Opengear gateway which will be the SSH server. You will need to make sure this file is in the correct format with the correct permissions with the following commands:

  *# dos2unix *
  */etc/config/users/testuser/.ssh/authorized_keys && chown testuser *
  /etc/config/users/testuser/.ssh/authorized_keys

- Using WinSCP copy the attached sshd_config over */etc/config/sshd_config* on the server (Makes sure public key authentication is enabled)
- Test the Public Key by logging in as "testuser" Test the Public Key by logging in as "testuser" to the client Opengear device and typing (you should not need to enter anything): # ssh -o StrictHostKeyChecking=no <server-ip>

To automate connection of the SSH tunnel from the client on every power-up you need to make the *clients /etc/config/rc.local* look like the following:

```
#!/bin/sh
ssh -L9001:127.0.0.1:4001 -N -o StrictHostKeyChecking=no testuser@<server-ip> &
```

This will run the tunnel redirecting local port 9001 to the server port 4001.

2.6.6 Fingerprinting

Fingerprints are used to ensure you are establishing an SSH session to who you think you are. On the first connection to a remote server you will receive a fingerprint which you can use on future connections.

This fingerprint is related to the host key of the remote server. Fingerprints are stored in *~/.ssh/known_hosts*.

To receive the fingerprint from the remote server, log in to the client as the required user (usually root) and establish a connection to the remote host:

```
# ssh remhost
    The authenticity of host 'remhost (192.168.0.1)' can't be established.
    RSA key fingerprint is 8d:11:e0:7e:8a:6f:ad:f1:94:0f:93:fc:7c:e6:ef:56.
    Are you sure you want to continue connecting (yes/no)?
```

At this stage, answer yes to accept the key. You should get the following message:

```
    Warning: Permanently added 'remhost,192.168.0.1' (RSA) to the list of
    known hosts.
```

You may be prompted for a password, but there is no need to log in - you have received the fingerprint and can Ctrl-C to cancel the connection.If the host key changes you will receive the following warning, and not be allowed to connect to the remote host:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@  WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@ IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Someone could be eavesdropping on you right now (man-in-the-middle attack)!

It is also possible that the RSA host key has just been changed.

The fingerprint for the RSA key sent by the remote host is

ab:7e:33:bd:85:50:5a:43:0b:e0:bd:43:3f:1c:a5:f8.

Please contact your system administrator.

Add correct host key in */.ssh/known_hosts* to get rid of this message.

Offending key in */.ssh/known_hosts:1*

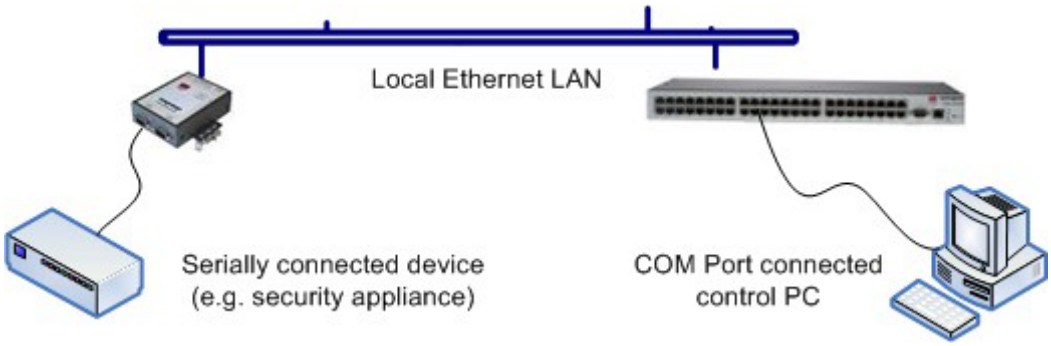
RSA host key for *remhost* has changed and you have requested strict checking.

Host key verification failed.

If the host key has been legitimately changed, it can be removed from the *~/.ssh/known_hosts* file and the new fingerprint added. If it has not changed, this indicates a serious problem that should be investigated immediately.

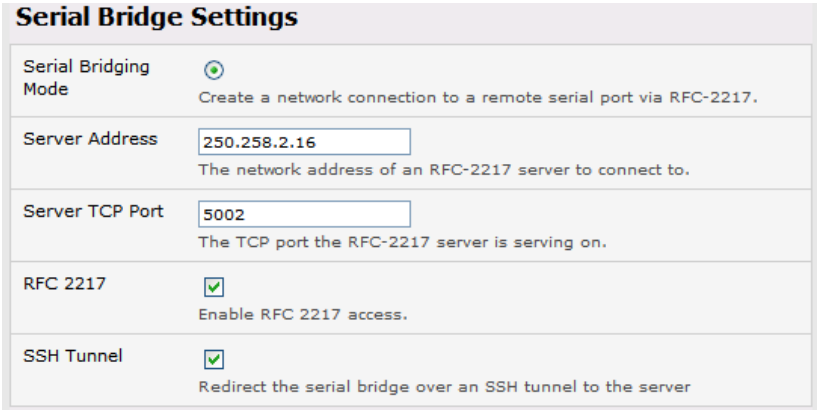
2.6.7 SSH tunneled serial bridging

You have the option to apply SSH tunneling when two console servers are configured for serial bridging.



The *Server* console server is setup in *Console Server* mode with either RAW or RFC2217 enabled and the *Client* console server is set up in Serial Bridging Mode with the Server Address, and Server TCP Port (4000 + port for RAW or 5000 + port # for RFC2217) specified:

- Select **SSH Tunnel** when configuring the **Serial Bridging Setting**



Next you will need to set up SSH keys for each end of the tunnel and upload these keys to the *Server* and *Client* console servers.

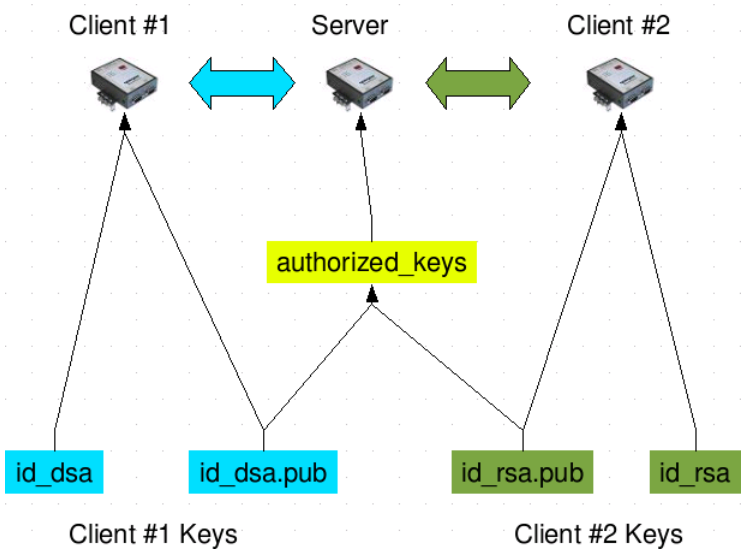
Client Keys:

The first step in setting up ssh tunnels is to generate keys. Ideally, you will use a separate, secure, machine to generate and store all keys to be used on the *console servers*. However, if this is not ideal to your situation, keys may be generated on the *console servers* themselves.

It is possible to generate only one set of keys, and reuse them for every SSH session. While this is not recommended, each organization will need to balance the security of separate keys against the additional administration they bring.

Generated keys may be one of two types - RSA or DSA (and it is beyond the scope of this document to recommend one over the other). RSA keys will go into the files *id_rsa* and *id_rsa.pub*. DSA keys will be stored in the files *id_dsa* and *id_dsa.pub*.

For simplicity going forward the term *private key* will be used to refer to either *id_rsa* or *id_dsa* and *public key* to refer to either *id_rsa.pub* or *id_dsa.pub.*



To generate the keys using OpenBSD's OpenSSH suite, we use the *ssh-keygen* program:

$ ssh-keygen -t [rsa|dsa]
Generating public/private [rsa|dsa] key pair.
Enter file in which to save the key *(/home/user/.ssh/id_[rsa|dsa]):*
Enter *passphrase* (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in */home/user/.ssh/id_[rsa|dsa].*
Your public key has been saved in */home/user/.ssh/id_[rsa|dsa].pub.*
The key fingerprint is:
28:aa:29:38:ba:40:f4:11:5e:3f:d4:fa:e5:36:14:d6 user@server
$

It is advisable to create a new directory to store your generated keys. It is also possible to name the files after the device they will be used for. For example:

$ mkdir keys
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): /*home/user/keys/control_room*
Enter *passphrase* (empty for no passphrase):
Enter same *passphrase* again:
Your identification has been saved in */home/user/keys/control_room*
Your public key has been saved in */home/user/keys/control_room.pub*.
The key fingerprint is:
28:aa:29:38:ba:40:f4:11:5e:3f:d4:fa:e5:36:14:d6 user@server
$

You should ensure there is no password associated with the keys. If there is a password, then the *console servers* will have no way to supply it as runtime.

Authorized Keys:

If the *console server* selected to be the server will only have one client device, then the *authorized_keys* file is simply a copy of the public key for that device. If one or more devices will be clients of the server, then the *authorized_keys* file will contain a copy of all of the public keys. RSA and DSA keys may be freely mixed in the *authorized_keys* file.

For example, assume we already have one server, called *bridge_server*, and two sets of keys, for the *control_room* and the *plant_entrance*:

$ ls /home/user/keys

control_room control_room.pub plant_entrance plant_entrance.pub

$ cat /home/user/keys/control_room.pub

/home/user/keys/plant_entrance.pub >

/home/user/keys/authorized_keys_bridge_server

Uploading Keys:

The keys for the server can be uploaded through the web interface, on the **System: Administration** page as detailed earlier. If only one client will be connecting, then simply upload the appropriate public key as the authorized keys file. Otherwise, upload the authorized keys file constructed in the previous step.

Each client will then need its own set of keys uploaded through the same page. Take care to ensure that the correct type of keys (DSA or RSA) goes in the correct spots, and that the public and private keys are in the correct spot.

2.7 Secure Sockets Layer (SSL) Support

Secure Sockets Layer (SSL) is a protocol developed by Netscape for transmitting private documents *via* the Internet. SSL works by using a private key to encrypt data that's transferred over the SSL connection.

The *console server* includes OpenSSL. The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

The OpenSSL toolkit is licensed under an Apache-style license, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions. In the *console server* OpenSSL is used primarily in conjunction with ‘http’ in order to have secure browser access to the GUI management console across insecure networks.

More documentation on OpenSSL is available from:

http://www.openssl.org/docs/apps/openssl.html

http://www.openssl.org/docs/HOWTO/certificates.txt

2.8 HTTPS

The Management Console UI is served using HTTPS by the built in *cherokee* webserver.

If your default network address is changed or the unit is to be accessed *via* a known Domain Name you can use the following steps to replace the default SSL Certificate and Private Key with ones tailored for your new address.

2.8.1 Generating an encryption key

To create a 1024 bit RSA key with a password issue the following command on the command line of a linux host with the *openssl* utility installed:

openssl genrsa -des3 -out ssl_key.pem 1024

2.8.2 Generating a self-signed certificate with OpenSSL

This example shows how to use OpenSSL to create a self-signed certificate. OpenSSL is available for most Linux distributions *via* the default package management mechanism. (Windows users can check *http://www.openssl.org/related/binaries.html*)

To create a 1024 bit RSA key and a self-signed certificate issue the following *openssl* command from the host you have *openssl* installed on:

openssl req -x509 -nodes -days 1000 \

-newkey rsa:1024 -keyout ssl_key.pem -out ssl_cert.pem

You will be prompted to enter a lot of information. Most of it doesn't matter, but the "Common Name" should be the domain name of your computer (*e.g.* test.opengear.com). When you have entered everything, the certificate will be created in a file called *ssl_cert.pem*.

2.8.3 Installing the key and certificate

The recommended method for copying files securely to the *console server* unit is with an SCP (Secure Copying Protocol) client. The *scp* utility is distributed with OpenSSH for most Unix distributions while Windows users can use something like the PSCP command line utility available with PuTTY.

The files created in the steps above can be installed remotely with the *scp* utility as follows:

scp ssl_key.pem root@<address of unit>:/etc/config/
scp ssl_cert.pem root@<address of unit>:/etc/config/

or using PSCP:

pscp -scp ssl_key.pem root@<address of unit>:/etc/config/
pscp -scp ssl_cert.pem root@<address of unit>:/etc/config/

PuTTY and the PSCP utility can be downloaded from:
*http://www.chiark.greenend.org.uk/~sgtatham/putty/download.htm*l

More detailed documentation on the PSCP can be found:
http://the.earth.li/~sgtatham/putty/0.58/htmldoc/Chapter5.html#pscp

2.8.4 Launching the HTTPS Server

Note that the easiest way to enable the HTTPS server is from the web Management Console. Simply click the appropriate checkbox in **Network: Services: HTTPS Server** and the HTTPS server will be activated (assuming the *ssl_key.pem* & *ssl_cert.pem* files exist in the */etc/config* directory).

Alternatively *inetd* can be configured to launch the secure *fnord* server from the command line of the unit as follows.

Edit the *inetd* configuration file. From the unit command line:

vi /etc/config/inetd.conf

Append a line:

443 stream tcp nowait root sslwrap -cert /etc/config/ssl_cert.pem -key /etc/config/ssl_key.pem -exec /bin/httpd /home/httpd"

Save the file and signal *inetd* of the configuration change.

kill -HUP `cat /var/run/inetd.pid`

The HTTPS server should be accessible from a web client at a URL similar to this: *https://<common name of unit>*

More detailed documentation about the *openssl* utility can be found at the website: *http://www.openssl.org/*

2.9 Power Strip Control

The *console server* supports a growing list of remote power-control devices (RPCs) which can be configured using the Management Console. These RPCs are controlled using the open source *PowerMan* and *Network UPS Tools* and with Opengear's *pmpower* utility.

2.9.1 The PowerMan tool

PowerMan provides power management in a data center or compute cluster environment. It performs operations such as power on, power off, and power cycle via remote power controller (RPC) devices.

Synopsis
powerman [-option] [targets]
pm [-option] [targets]

Options
-1, --on Power ON targets.
-0, --off Power OFF targets.
-c, --cycle Power cycle targets.
-r, --reset Assert hardware reset for targets (if implemented by RPC).
-f, --flash Turn beacon ON for targets (if implemented by RPC).
-u, --unflash Turn beacon OFF for targets (if implemented by RPC).
-l, --list List available targets. If possible, output will be compressed into a host range (see TARGET SPECIFICATION below).
-q, --query Query plug status of targets. If none specified, query all targets. Status is not cached; each time this option is used, powerman queries the appropriate RPC's. Targets connected to RPC's that could not be contacted (e.g. due to network failure) are reported as status "unknown". If possible, output will be compressed into host ranges.
-n, --node Query node power status of targets (if implemented by RPC). If no targets specified, query all targets. In this context, a node in the OFF state could be ON at the plug but operating in standby power mode.

-b, --beacon Query beacon status (if implemented by RPC). If no targets are specified, query all targets.
-t, --temp Query node temperature (if implemented by RPC). If no targets are specified, query all targets. Temperature information is not interpreted by powerman and is reported as received from the RPC on one line per target, prefixed by target name.
-h, --help Display option summary.
-L, --license Show powerman license information.
-d, --destination host[:port] Connect to a powerman daemon on non-default host and optionally port.
-V, --version Display the powerman version number and exit.
-D, --device Displays RPC status information. If targets are specified, only RPC's matching the target list is displayed.
-T, --telemetry Causes RPC telemetry information to be displayed as commands are processed. Useful for debugging device scripts.
-x, --exprange Expand host ranges in query responses.

For more details refer *http://linux.die.net/man/1/powerman*
Also refer *powermand* (*http://linux.die.net/man/1/powermand)* documentation and *powerman.conf* (*http://linux.die.net/man/5/powerman.conf*)

Target Specification
powerman target hostnames may be specified as comma separated or space separated hostnames or host ranges. Host ranges are of the general form: prefix[n-m,l-k,...], where n < m and l < k, etc., This form should not be confused with regular expression character classes (also denoted by "[]"). For example, foo[19] does not represent foo1 or foo9, but rather represents a degenerate range: foo19.

This range syntax is meant only as a convenience on clusters with a prefix NN naming convention and specification of ranges should not be considered necessary -- the list foo1,foo9 could be specified as such, or by the range foo[1,9].

Some examples of powerman targets follow.

Power on hosts bar,baz,foo01,foo02,...,foo05: *powerman --on bar baz foo[01-05]*

Power on hosts bar,foo7,foo9,foo10: *powerman --on bar,foo[7,9-10]*

Power on foo0,foo4,foo5: *powerman --on foo[0,4-5]*

As a reminder to the reader, some shells will interpret brackets ([and]) for pattern matching. Depending on your shell, it may be necessary to enclose ranged lists within quotes. For example, in tcsh, the last example above should be executed as:

powerman --on "foo[0,4-5]"

2.9.2 The pmpower tool

The *pmpower* utility is a high level tool for manipulating remote preconfigured power devices connected to the *console server* either via a serial or network connection. The PDU UPS and IPMI power devices are variously controlled using the open source *PowerMan, IPMItool* or *Network UPS Tools* and Opengear's *pmpower* utility arches over these tools so the devices can be controlled through the one command line:

pmpower [-?h] [-l device | -r host] [-o outlet] [-u username] [-p password] action

-?/-h This help message.
-l The serial port to use.
-o The outlet on the power target to apply to
-r The remote host address for the power target
-u Override the configured username
-p Override the configured password
on This *action* switches the specified device or outlet(s) on

off This *action* switches the specified device or outlet(s) off
cycle This *action* switches the specified device or outlet(s) off and on again
status This *action* retrieves the current status of the device or outlet

Examples:

To turn outlet 4 of the power device connected to serial port 2 on: *# pmpower -l port02 -o 4 on*

To turn an IPMI device off located at IP address 192.168.1.100 (where username is 'root' and password is 'calvin': *# pmpower -r 192.168.1.100 -u root -p calvin off*

Default system Power Device actions are specified in */etc/powerstrips.xml*. Custom Power Devices can be added in */etc/config/powerstrips.xml*. If an action is attempted which has not been configured for a specific Power Device *pmpower* will exit with an error.

2.9.3 Adding new RPC devices

There are a number of simple paths to adding support for new RPC devices.

The first is to have scripts to support the particular RPC included in either the open source *PowerMan* project *(http://sourceforge.net/projects/powerman)* or the open source *NUT UPS Tools* project. The *PowerMan* device specifications are rather weird and it is suggested that you leave the actual writing of these scripts to the PowerMan authors. However documentation on how they work can be found at *http://linux.die.net/man/5/powerman.dev*. The *Network UPS Tools(NUT)* project has recently moved on from its UPS management origins to also cover SNMP PDUs (and embrace PowerMan). Opengear progressively includes the updated *PowerMan* and *NUT* build into the *console server* firmware releases.

The second path is to directly add support for the new RPC devices (or to customize the existing RPC device support) on your particular *console server*. The **Manage: Power** page uses information contained in */etc/powerstrips.xml* to configure and control devices attached to a serial port. The configuration also looks for (and loads) */etc/config/powerstrips.xml* if it exists.

The user can add their own support for more devices by putting definitions for them into */etc/config/powerstrips.xml*. This file can be created on a host system and copied to the Management Console device using *scp*. Alternatively, login to the Management Console and use *ftp* or *wget* to transfer files.

Here is a brief description of the elements of the XML entries in */etc/config/powerstrips.xml*.

```
<powerstrip>
        <id>Name or ID of the device support</id>
        <outlet port="port-id-1">Display Port 1 in menu</outlet>
        <outlet port="port-id-2">Display Port 2 in menu</outlet>
        ...
        <on>script to turn power on</on>
        <off>script to power off</off>
        <cycle>script to cycle power</cycle>
        <status>script to write power status to /var/run/power-status</status>
        <speed>baud rate</speed>
        <charsize>character size</charsize>
        <stop>stop bits</stop>
        <parity>parity setting</parity>
</powerstrip>
```

The *id* appears on the web page in the list of available devices types to configure.

The outlets describe targets that the scripts can control. For example a power control board may control several different outlets. The port-id is the native name for identifying the outlet. This value will be passed to the scripts in the environment variable *outlet*, allowing the script to address the correct outlet.

There are four possible scripts: *on, off, cycle* and *status*.

When a script is run, it's standard input and output is redirected to the appropriate serial port. The script receives the outlet and port in the *outlet* and *port* environment variables respectively.

The script can be anything that can be executed within the shell.

All of the existing scripts in */etc/powerstrips.xml* use the *pmchat* utility.

pmchat works just like the standard unix "chat" program, only it ensures interoperation with the port manager.

The final options, *speed, charsize, stop* and *parity* define the recommended or default settings for the attached device.

2.10 IPMItool

The *console server* includes the *ipmitool* utility for managing and configuring devices that support the Intelligent Platform Management Interface (IPMI) version 1.5 and version 2.0 specifications.

IPMI is an open standard for monitoring, logging, recovery, inventory, and control of hardware that is implemented independent of the main CPU, BIOS, and OS. The service processor (or Baseboard Management Controller, BMC) is the brain behind platform management and its primary purpose is to handle the autonomous sensor monitoring and event logging features.

The *ipmitool* program provides a simple command-line interface to this BMC. It features the ability to read the sensor data repository (SDR) and print sensor values, display the contents of the System Event Log (SEL), print Field Replaceable Unit (FRU) inventory information, read and set LAN configuration parameters, and perform remote chassis power control.

SYNOPSIS

ipmitool [**-c**|**-h**|**-v**|**-V**] **-I** *open* <*command*>

ipmitool [**-c**|**-h**|**-v**|**-V**] **-I** *lan* **-H** <*hostname*>
[**-p** <*port*>]
[**-U** <*username*>]
[**-A** <*authtype*>]
[**-L** <*privlvl*>]
[**-a**|**-E**|**-P**|**-f** <*password*>]
[**-o** <*oemtype*>]
<*command*>

ipmitool [**-c**|**-h**|**-v**|**-V**] **-I** *lanplus* **-H** <*hostname*>
[**-p** <*port*>]
[**-U** <*username*>]
[**-L** <*privlvl*>]
[**-a**|**-E**|**-P**|**-f** <*password*>]
[**-o** <*oemtype*>]
[**-C** <*ciphersuite*>]
<*command*>

DESCRIPTION

This program lets you manage Intelligent Platform Management Interface (IPMI) functions of either the local system, via a kernel device driver, or a remote system, using IPMI V1.5 and IPMI v2.0. These functions include printing FRU information, LAN configuration, sensor readings, and remote chassis power control.

IPMI management of a local system interface requires a compatible IPMI kernel driver to be installed and configured. On Linux this driver is called *OpenIPMI* and it is included in standard distributions. On Solaris this driver is called *BMC* and is included in Solaris 10. Management of a remote station requires the IPMI-over-LAN interface to be enabled and configured. Depending on the particular requirements of each system it may be possible to enable the LAN interface using *ipmitool* over the system interface.

OPTIONS

-a Prompt for the remote server password.

-A <*authtype*>
Specify an authentication type to use during IPMIv1.5 *lan* session activation. Supported types are NONE, PASSWORD, MD5, or OEM.

-c Present output in CSV (comma separated variable) format. This is not available with all commands.

-C <*ciphersuite*>
The remote server authentication, integrity, and encryption algorithms to use for IPMIv2 *lanplus* connections. See table 22-19 in the IPMIv2 specification. The default is 3 which specifies RAKP-HMAC-SHA1 authentication, HMAC-SHA1-96 integrity, and AES-CBC-128 encryption algorithms.

-E The remote server password is specified by the environment variable *IPMI_PASSWORD*.

-f <*password_file*>
Specifies a file containing the remote server password. If this option is absent, or if password_file is empty, the password will default to NULL.

-h Get basic usage help from the command line.

-H <*address*>
Remote server address, can be IP address or hostname. This option is required for *lan* and *lanplus* interfaces.

-I <*interface*>
Selects IPMI interface to use. Supported interfaces that are compiled in are visible in the usage help output.

-L <*privlvl*>
Force session privilege level. Can be CALLBACK, USER, OPERATOR, and ADMIN. Default is ADMIN.

-m <*local_address*>
Set the local IPMB address. The default is 0x20 and there should be no need to change it for normal operation.

-o <*oemtype*>
Select OEM type to support. This usually involves minor hacks in place in the code to work around quirks in various BMCs from various manufacturers. Use *-o list* to see a list of current supported OEM types.

-p <*port*>
Remote server UDP port to connect to. Default is 623.

-P <*password*>
Remote server password is specified on the command line. If supported it will be obscured in the process list. **Note!** Specifying the password as a command line option is not recommended.

-t <*target_address*>
Bridge IPMI requests to the remote target address.

-U <*username*>
Remote server username, default is NULL user.

-v Increase verbose output level. This option may be specified multiple times to increase the level of debug output. If given three times you will get hexdumps of all incoming and outgoing packets.

-V Display version information.

If no password method is specified then *ipmitool* will prompt the user for a password. If no password is entered at the prompt, the remote server password will default to NULL.

SECURITY

The *ipmitool* documentation highlights that there are several security issues to be considered before enabling the IPMI LAN interface. A remote station has the ability to control a system's power state as well as being able to gather certain platform information. To reduce vulnerability it is strongly advised that the IPMI LAN interface only be enabled in 'trusted' environments where system security is not an issue or where there is a dedicated secure 'management network' or access has been provided through an *console server*.

Further it is strongly advised that you should not enable IPMI for remote access without setting a password, and that that password should not be the same as any other password on that system.

When an IPMI password is changed on a remote machine with the IPMIv1.5 *lan* interface the new password is sent across the network as clear text. This could be observed and then used to attack the remote system. It is thus recommended that IPMI password management only be done over IPMIv2.0 *lanplus* interface or the system interface on the local station.

For IPMI v1.5, the maximum password length is 16 characters. Passwords longer than 16 characters will be truncated.

For IPMI v2.0, the maximum password length is 20 characters; longer passwords are truncated.

COMMANDS

help

This can be used to get command-line help on *ipmitool* commands. It may also be placed at the end of commands to get option usage help.

ipmitool help

Commands:
raw Send a RAW IPMI request and print response
lan Configure LAN Channels
chassis Get chassis status and set power state
event Send pre-defined events to MC
mc Management Controller status and global enables
sdr Print Sensor Data Repository entries and readings
sensor Print detailed sensor information
fru Print built-in FRU and scan SDR for FRU locators
sel Print System Event Log (SEL)
pef Configure Platform Event Filtering (PEF)
sol Configure IPMIv2.0 Serial-over-LAN
isol Configure IPMIv1.5 Serial-over-LAN
user Configure Management Controller users
channel Configure Management Controller channels
session Print session information
exec Run list of commands from file
set Set runtime variable for shell and exec

ipmitool chassis help

Chassis Commands: status, power, identify, policy, restart_cause, poh, bootdev

ipmitool chassis power help

chassis power Commands: status, on, off, cycle, reset, diag, soft

You will find more details on *ipmitools* at http://*ipmitool*.sourceforge.net/manpage.html.

2.11 REST API

The latest Console Server REST API is located here: https://ftp.opengear.com/download/api/cs/.

The current endpoints include:

/sessions - authenticate the user and create a session token for accessing all other Lighthouse endpoints.

/registration - used by Lighthouse to provide the Node with the address and credentials to retrieve an enrollment package for the device

/auth - update and retrieve the authentication configuration from the console server

/groups - update and retrieve the group configuration from the console server

/users – get user list configuration for console server

/serialPorts - retrieve the serial port configuration from the console server

/secureShell - get the SSH port of the device

/system/version - retrieve the system version from the console server

/interfaces - list network interfaces

/interfaces/cellmodem/status - retrieve status data about the cellular modem in the device

/interfaces/cellmodem/tests - list available cell modem tests

/interfaces/cellmodem/tests/ping - test ping on a cellmodem interface

/interfaces/cellmodem/tests/http - test ping on a cellmodem interface

/interfaces/cellmodem/tests/dns - test ping on a cellmodem interface

/backup – request a system configuration backup file. This is used by Lighthouse to create a node backup and store it on Lighthouse.

2.12 Scripts for Managing Secondary Nodes

When the *console servers* are cascaded the Primary is in control of the serial ports on the Secondaries, and the Primary's Management Console provides a consolidated view of the settings for its own and all the Secondary's serial ports. However, the Primary does not provide a fully consolidated view e.g. *Status: Active Users* only displays those users active on the Primary's ports and you will need to write a custom bash script that parses the port logs if you want to find out who's logged in to cascaded serial ports from the primary.

You will probably also want to enable remote or USB logging, as local logs only buffer 8K of data and don't persist between reboots.

This script would e.g. parse each port log file line by line, each time it sees *'LOGIN: username'*, it adds username to the list of connected users for that port, each time it sees *'LOGOUT: username'* it removes it from the list. The list can then be nicely formatted and displayed. It's also possible to run this as a CGI script on the remote log server.

To enable log storage and connection logging:
- Select *Alerts & Logging: Port Log*
- *Configure* log storage
- Select *Serial & Network: Serial Port*, *Edit* the serial port(s)
- Under *Console server*, select *Logging Level 1* and click Apply

There's a useful tutorial on creating a bash script CGI at
http://www.yolinux.com/TUTORIALS/LinuxTutorialCgiShellScript.html

Similarly the Primary does maintain a view of the status of the secondaries:
- Select *Status: Support Report*
- Scroll down to *Processes*
- Look for: */bin/ssh -MN -o ControlPath=/var/run/cascade/%h nodename*
- These are the nodes that are connected
- Note the end of the secondaries' names will be truncated, so the first 5 characters must be unique

Alternatively, you can write a custom CGI script as described above. The currently connected nodes can be determined by running: *ls /var/run/cascade* and the configured nodes can be displayed by running: *config -g config.cascade.nodes*

2.13 SMS Server Tools

Firmware releases V3.1 and later include the *SMS Server Tools software* which provides an SMS Gateway which can send and receive short messages through GSM modems and mobile phones.

You can send short messages by simply storing text files into a special spool directory. The program monitors this directory and sends new files automatically. It also stores received short messages into another directory as text files. Binary messages (including Unicode text) are also supported, for example ring tone messages. It's also possible to send a WAP Push message to the WAP / MMS capable mobile phone.

The program can be run as a SMS daemon which can be started automatically when the operating system starts. High availability can be ensured by using multiple GSM devices (currently up to 64, this limit is easily changeable).

The program can run other external programs or scripts after events like reception of a new message, successful sending and also when the program detects a problem. These programs can inspect the related text files and perform automatic actions

The SMS Server Tools software needs a GSM modem (or mobile phone) with SMS command set according to the European specifications GSM 07.05 (=ETSI TS 300 585) and GSM 03.38 (=ETSI TS 100 900). AT command set is supported. Devices can be connected with serial port, infrared or USB.

For more information refer *http://smstools3.kekekasvi.com* or the online Opengear *faq.html*

2.14 Multicast

By default, all Opengear console servers come with Multicasting enabled. Multicasting provides Opengear products with the ability to simultaneously transmit information from a single device to a select group of hosts.

Multicasting can be disabled and re-enabled from the command line (Firmware releases V3.1 and later). To disable multicasting type:

ifconfig eth0 -multicast

To re-enable multicasting from the command line type:

ifconfig eth0 multicast

IPv6 may need to be restarted when toggling between multicast states.

2.15 Bulk Provisioning

Opengear appliances include wizard scripts to facilitate configuration and deployment en masse. These wizards operate at the command line level, so knowledge of the Linux command line and shell scripting is useful, but not necessary – they aim to be user-friendly enough for remote hands to manage. This *bulk provisioning* feature is supported by firmware version 3.9.1 or later, and Lighthouse version 4.4.0 and later (optional).

Both the bulk provisioning of Opengear appliances and bulk enrollment of these appliances into Lighthouse central management system(s) is supported. These features may be used separately or in conjunction.

Using this method, an Opengear appliance can be fully configured and enrolled into Lighthouse with minimal interaction, in under 5 minutes. The basic steps are:

1. Configure an individual "*golden primary*" appliance with the baseline configuration shared by all Opengear appliances. This may be a minimal configuration if the installs are quite diverse, or a complete configuration when dealing with replicated installs.

2. Use make-template to turn the golden primary's active configuration into a template configuration that may be applied to other appliances.

3. Create an OPG backup of the templated golden primary appliance.
4. Restore this configuration to each target devices via the CLI, web UI or using a USB thumb drive.

5. Login via the CLI to complete configuration using *setup-wizard*.

6. (Optional) On Lighthouse, use *enrollment-wizard* to automatically place appliances under management. This may be local/routable appliances, or remote appliances that have automatically Call Home using *callhome-wizard*.

Note: Full details for the above steps can be found in the Knowledge Base

2.16 Zero Touch Provisioning

Zero Touch Provisioning (ZTP) was introduced with firmware release 3.15.1 to allow Opengear appliances to be provisioned during their initial boot from a DHCP server.

2.16.1 Preparation

These are typical steps for configuration over a trusted network:

1. Configure a same-model Opengear device.
2. Optionally use the *Bulk Provisioning* wizard scripts to remove any appliance-specific settings (i.e. create a template configuration) and/or prepare the configuration for automated Lighthouse enrollment.
3. Save the configuration as an Opengear backup (.opg) file under *System: Configuration Backup* in the web UI, or via *config -e* in the CLI. Alternatively, you can save the XML configuration as a file ending in .xml.
4. Publish the .opg or.xml file on a fileserver that understands one of the HTTPS, HTTP, FTP or TFTP protocols.
5. Configure your DHCP server to include a "vendor specific" option for Opengear devices. The option text should be a URL to the location of the .opg or .xml file. The option text should not exceed 250 characters in length. It must end in either .opg or .xml.
6. Connect a new Opengear device (either at defaults from the factory, or config erased) to the network and apply power.
7. It may take up to 5 minutes for the device to find the .opg or .xml file via DHCP, download, install the file and reboot itself.

2.16.2 Example ISC DHCP server configuration

The following is an example of an ISC DHCP server configuration fragment for serving an .opg configuration image:

```
option space opengear code width 1 length width 1;
option opengear.config-url code 1 = text;


class "opengear-ztp" {
 match if option vendor-class-identifier ~~ "^Opengear/";
 vendor-option-space opengear;
 option opengear.config-url "https://example.com/opg/${class}.opg";
    }
```

For other DHCP servers, please consult their documentation on specifying "Vendor Specific" option fields. We use sub-option 1 to hold the URL text.

2.16.3 Setup for an untrusted LAN

If network security is a concern, and you can have remote hands insert a trusted USB flash drive into the Opengear device during provisioning, then follows are a summary of the steps required for deploying configuration in an untrusted network:

1. Generate an X.509 certificate for the client. Place it and its private key file onto a USB flash drive (concatenated as a single file, client.pem).
2. Set up a HTTPS server that restricts access to the .opg or .xml file for HTTPS onnections providing the client certificate.

3. Put a copy of the CA cert (that signed the HTTP server's certificate) onto the USB flash drive as well (ca-bundle.crt).
4. Insert the USB flash drive into the Opengear device **before attaching power or network**.
5. Continue with the steps above but using only a https URL.
6. A detailed step-by-step document for preparing a USB flash drive and using OpenSSL to create keys is at Howto: set up a USB key for authenticated restore

2.16.4 How it works

This section explains in detail how the Opengear device uses DHCP to obtain its initial configuration.

First, an Opengear console manager is either configured or unconfigured. ZTP needs it to be in an unconfigured state, which is only obtained in the following ways:

- Firmware programming at factory
- Pressing the Config Erase button twice during operation
- Selecting *Config Erase* under *System: Administration* in the web UI, and rebooting
- Creating the file /etc/config/.init and then rebooting (command-line)

When an unconfigured Opengear boots, it performs these steps to find a configuration:

- The Opengear device transmits a DHCP DISCOVER request onto its primary Network Interface (wan). This DHCP request will carry a Vendor Class Identifier of the form Opengear/model-name (for example, Opengear/ACM5003-M) and its parameter request list will include option 43 (Vendor-Specific Information).
- On receipt of a DHCP OFFER, the device will use the information in the offer to assign an IPv4 address to its primary Network Interface, add a default route, and prepare its DNS resolver.
- If the offer also contained an option 43 with sub-option 1, the device interprets the sub-option as a whitespace-separated list of URLs to configuration files to try to restore.
- If an NTP server option was provided in the DHCP offer, the system clock is (quickly) synchronized with the NTP server.
- The system now searches all attached USB storage devices for two optional certificate files. The first file is named ca-bundle.crt and the second one is whichever one of the following filenames is found first:
    - client-AABBCCDDEEFF.pem (where AABBCCDDEEFF is the MAC address of the primary network interace); or

  - client-MODEL.pem (where MODEL is the (vendor class) model name in lowercase, truncated to before the first hyphen); or
  - client.pem
- If both files are found (ca-bundle.crt and a client.pem), then secure mode is enabled for the next section.
- Each URL in the list obtained from option 43 sub-option 1 is tried in sequence until one succeeds:
  - The URL undergoes substring replacement from the following table:

Substring	Replaced by	Example
${mac}	the 12-digit MAC address of the device, lowercase	0013b600b669
${model}	the full model name, in lowercase	acm5504-5-g-w-i
${class}	the firmware hardware class	ACM550x
${version}	the firmware version number	3.15.1

  - The resulting URL must end in .opg or .xml (an optional ?query-string is permitted). It is doesn't, then it is skipped and the next URL is tried.
  - In secure mode, the URL must use the https scheme or it is skipped.
  - Otherwise the available schemes are: http https tftp ftp ftps
  - The curl program is used to download the URL.
  - In secure mode, the server's certificate must validate against the ca-bundle.crt. The (reqiured) client.pem file is provided to authenticate the client to the server. Please see the curl documentation for the format of these files.
- The URL is downloaded. For .opg files its header is checked to see if it is compatible with the current device. For .xml files, a parse check is made. If the check fails, the downloaded file is abandoned and the next URL is tried.
- The file is imported into the current configuration.
- The system checks to see if a hostname has been set in the config. If not, it is set to ${model}-${mac}.
- The system checks to see if it is still in an unconfigured state. If it is, then the network interface mode is set to DHCP. This effectively forces the system into a configured state, preventing a future reboot loop.
- The system reboots

If all the URLs were skipped or failed, the system will wait for 30 seconds before retrying again. It will retry all the URLs up to 10 times. After the 10th retry, the system reboots. If the system has been manually configured in the meantime, the retries stop and ZTP is disabled.

If no option 43 is received over DHCP, no URLs are downloaded and no reboots occur: the system must be manually configured. Once configured (manually or by ZTP), an Opengear will no longer request option 43 from the DHCP server, and it will ignore any option 43 configuration URLs presented to it.

2.17 Internal Storage

Some models have an internal USB flash drive, a non-volatile **NAND** flash partition, or both, which can be used by portmanager for log storage and the TFTP/FTP server for file storage.

These storage devices are automatically mounted as subdirectories of /var/mnt/. The default directory served by FTP or TFTP is set to the preferred internal storage (if any), otherwise the first detected attached USB storage. The location of portmanager logs must be manually configured.

2.17.1 Filesystem location of FTP/TFTP directory

Product	Preferred storage	Directory
ACM7000	Internal flash	/var/mnt/storage.nvlog/tftpboot/
CM7100	Internal USB flash	/var/mnt/storage.usb/tftpboot/
IM7200	Internal USB flash	/var/mnt/storage.usb/tftpboot/
ACM5500	Internal USB flash	/var/mnt/storage.usb/tftpboot/
ACM5000-F	Internal USB flash option	/var/mnt/storage.usb/tftpboot/
Other products with USB	First-attached USB storage	/var/mnt/storage.usb/tftpboot/

2.17.2 Filesystem location of portmanager logs

Port log server type	Directory
USB Flash Memory	/var/mnt/storage.usb/
Non-volatile internal storage	/var/mnt/storage.nvlog/
MicroSD Card	/var/mnt/storage.sd/
Other (NFS, CIFS, etc.)	*As explicitly configured*

2.17.3 Configuring FTP/TFTP directory

The FTP or TFTP services can be configured to serve different directories via the command line, e.g.:

```
config -s config.services.ftp.directory=/var/mnt/storage.usb/my-ftp-dir
config -r services
```

The directory will be created if it doesn't already exist.

2.18.3 Mounting a preferred USB disk by label

Currently, the "first" USB storage device is mounted at /var/mnt/storage.usb by detecting the lowest numbered disk partition, e.g. `/dev/sda1`, but this can be constrained to match a particular port or a labelled device.

1. Attach the USB disk you plan to use

2. Look in directories `/dev/disk/by-path/` or `/dev/disk/by-label/` to find a suitably stable way of identifying your disk
3. Use the following command to see the current device matching string used:

   ```
   config -g config.storage.usb.device
   ```

4. Change the path match with (for example):

   ```
   config -s config.storage.usb.device=/dev/disk/by-label/1103
   ```

APPENDIX A: Linux Commands & Source Code

The *console server* platform is a dedicated Linux computer, optimized to provide monitoring and secure access to serial and network consoles of critical server systems and their supporting power and networking infrastructure.

Opengear *console servers* are built on the uCLinux distribution as developed by the uCLinux project. This is GPL code and source can be found at *http://cvs.uclinux.org*.

Some uCLinux commands have config files that can be altered (e.g. *portmanager, inetd*, *init*, *sshd).*

Other commands you can run and do neat stuff with (e.g. *loopback*, *bash (shell)*, *ftp, hwclock*, *iproute*, *iptables*, *netcat*, *ifconfig*, *mii-tool*, *netstat*, *route*, *ping*, *portmap*, *pppd*, *routed*, *setserial*, *smtpclient*, *stty*, *stunel*, *tcpdump*, *tftp*, *tip*, *traceroute)*

Below are most of the standard uCLinux and Busybox commands (and some custom Opengear commands) that are in the default build tree. The *Administrator* can use these to configure the *console server*, and monitor and manage attached serial console and host devices:

WARNING: *smbmount* is deprecated and not maintained any longer. mount.cifs (mount -t cifs) should be used instead of *smbmount*..

addgroup *	Add a group or add a user to a group
adduser *	Add a user
agetty	alternative Linux getty
arp	Manipulate the system ARP cache
arping	Send ARP requests/replies
bash	GNU Bourne-Again Shell
busybox	Swiss army knife of embedded Linux commands
cat *	Concatenate FILE(s) and print them to stdout
chat	Useful for interacting with a modem connected to stdin/stdout
chgrp *	Change file access permissions
chmod *	Change file access permissions
chown *	Change file owner and group
config	Opengear tool to manipulate and query the system configuration from the command line
cp *	Copy files and directories
date *	Print or set the system date and time
dd *	Convert and copy a file
deluser *	Delete USER from the system
df *	Report file system disk space usage
dhcpd	Dynamic Host Configuration Protocol server
discard	Network utility that listens on the discard port
dmesg *	Print or control the kernel ring buffer
echo *	Print the specified ARGs to stdout
erase	Tool for erasing MTD partitions
eraseall	Tool for erasing entire MTD partitions
expect	Waits for user input
false *	Do nothing, unsuccessful
find	Search for files
Flashw	Write data to individual flash devices
flatfsd	Daemon to save RAM file systems back to FLASH
ftp	Internet file transfer program
gen-keys	SSH key generation program

getopt *	Parses command options
gettyd	Getty daemon
grep *	Print lines matching a pattern
gunzip *	Compress or expand files
gzip *	Compress or expand files
hd	ASCII, decimal, hexadecimal, octal dump
hostname *	Get or set hostname or DNS domain name
httpd	Listen for incoming HTTP requests
hwclock	Query and set hardware clock (RTC)
inetd	Network super-server daemon
inetd-echo	Network echo utility
init	Process control initialization
ip	Show or manipulate routing, devices, policy routing and tunnels
ipmitool	Linux IPMI manager
iptables	Administration tool for IPv4 packet filtering and NAT
ip6tables	Administration tool for IPv6 packet filtering
iptables-restore	Restore IP Tables
iptables-save	Save IP Tables
kill *	Send a signal to a process to end gracefully
ln *	Make links between files
login	Begin session on the system
loopback	Opengear loopback diagnostic command
loopback1	Opengear loopback diagnostic command
loopback2	Opengear loopback diagnostic command
loopback8	Opengear loopback diagnostic command
loopback16	Opengear loopback diagnostic command
loopback48	Opengear loopback diagnostic command
ls *	List directory contents
mail	Send and receive mail
mkdir *	Make directories
mkfs.jffs2	Create an MS-DOS file system under Linux
mknod *	Make block or character special files
more *	File perusal filter for crt viewing
mount *	Mount a file system
msmtp	SMTP mail client
mv *	Move (rename) files
nc	TCP/IP Swiss army knife
netflash	Upgrade firmware on ucLinux platforms using the blkmem interface
netstat	Print network connections, routing tables, interface statistics etc
ntpd	Network Time Protocol (NTP) daemon
pgrep	Display process(es) selected by regex pattern
pidof	Find the process ID of a running program
ping	Send ICMP ECHO_REQUEST packets to network hosts
ping6	IPv6 ping
pkill	Sends a signal to process(es) selected by regex pattern
pmchat	Opengear command similar to the standard chat command (via portmanager)
pmdeny	

pminetd	
pmloggerd	
pmshell	Opengear command similar to the standard *tip* or *cu* but all serial port access is directed via the portmanager.
pmusers	Opengear command to query portmanager for active user sessions
portmanager	Opengear command that handles all serial port access
portmap	DARPA port to RPC program number mapper
pppd	Point-to-Point protocol daemon
ps *	Report a snapshot of the current processes
pwd *	Print name of current/working directory
reboot *	*Soft* reboot
rm *	Remove files or directories
rmdir *	Remove empty directories
routed	Show or manipulate the IP routing table
routed	Show or manipulate the IP routing table
routef	IP Route tool to flush IPv4 routes
routel	IP Route tool to list routes
scp	Secure copy (remote file copy program)
sed *	Text stream editor
setmac	Sets the MAC address
setserial	Sets and reports serial port configuration
sh	Shell
showmac	Shows MAC address
sleep *	Delay for a specified amount of time
snmpd	SNMP daemon
snmptrap	Sends an SNMP notification to a manager
sredird	RFC 2217 compliant serial port redirector
ssh	OpenSSH SSH client (remote login program)
ssh-keygen	Authentication key generation, management, and conversion
sshd	OpenSSH SSH daemon
stty	Change and print terminal line settings
stunnel	Universal SSL tunnel
sync *	Flush file system buffers
sysctl	Configure kernel parameters at runtime
syslogd	System logging utility
tar *	The tar archiving utility
tc	Show traffic control settings
tcpdump	Dump traffic on a network
telnetd	Telnet protocol server
tftp	Client to transfer a file from/to tftp server
tftpd	Trivial file Transfer Protocol (tftp) server
tip	Simple terminal emulator/cu program for connecting to modems and serial devices

top	Provide a view of process activity in real time
touch *	Change file timestamps
traceroute	Print the route packets take to network host
traceroute6	Traceroute for IPv6
true *	Returns an exit code of TRUE (0)
umount *	Unmounts file systems
uname *	Print system information
usleep *	Delay for a specified amount of time
vconfig *	Create and remove virtual Ethernet devices
vi *	Busybox clone of the VI text editor
w	Show who is logged on and what they are doing
zcat *	Identical to gunzip -c

Commands above which are appended with '*' come from Busybox (the Swiss Army Knife of embedded Linux*) http://www.busybox.net/downloads/BusyBox.html*.

Others are generic Linux commands and most commands the ***-h*** or ***--help*** argument to provide a terse runtime description of their behavior. More details on the generic Linux commands can found online at *http://en.tldp.org/HOWTO/HOWTO-INDEX/howtos.html* and *http://www.faqs.org/docs/Linux-HOWTO/Remote-Serial-Console-HOWTO.html*

An updated list of the commands in the latest *console server* build can be found at *http://www.opengear.com/faq233.html*. However it may be worth using ***ls*** command to view all the commands actually available in the */bin* directory in your *console server*.

There are a number of Opengear tools that make it simple to configure the *console server* and ensure the changes are stored in the *console server*'s flash memory etc. These include:

- **config** which allows manipulation and querying of the system configuration from the command line. With *config* a new configuration can be activated by running the relevant configurator, which performs the action necessary to make the configuration changes live
- **portmanager** which provides a buffered interface to each serial port. It is supported by the pmchat and pmshell commands which ensure all serial port access is directed via the *portmanager*
- **pmpower** is a configurable tool for manipulating remote power devices that are serially or network connected to the *console server*

There are also a number of other CLI commands related to other open source tools embedded in the *console server* including:

- **PowerMan** provides power management for many preconfigured remote power controller (RPC) devices. For CLI details refer *http://linux.die.net/man/1/powerman*
- **Network UPS Tools (NUT)** provides reliable monitoring of UPS and PDU hardware and ensure safe shutdowns of the systems which are connected - with a goal to monitor every kind of UPS and PDU. For CLI details refer *http://www.networkupstools.org*
- **Nagios** is a popular enterprise-class management tool that provides central monitoring of the hosts and services in distributed networks. For CLI details refer http://www.nagios.org

Many components of the *console server* software are licensed under the GNU General Public License (version 2), which Opengear supports. You may obtain a copy of the GNU General Public License at

http://www.fsf.org/copyleft/gpl.html. Opengear will provide source code for any of the components of the software licensed under the GNU General Public License upon request.

Note: The software included in each Opengear console server contains copyrighted software that is licensed under the GPL (refer Appendix F for a copy of the GPL license). You may obtain the latest snapshot source code package on a CD by sending a money order or check for $5 to:
Opengear Support
630 West 9560 South, Suite A
Sandy, UT 84070, USA

Alternately the complete source code corresponding to each released version is available from us for a period of three years after its last shipment. If you would like the source code for an earlier release than the latest current release please write “source for firmware Version x.xx ” in the memo line of your payment.

This offer is valid to anyone in receipt of this information.

The *console server* also embodies the *okvm* console management software. This is GPL code and the full source is available from *http://okvm.sourceforge.net.*

The *console server* BIOS (boot loader code) is a port of *uboot* which is also a GPL package with source openly available.

The *console server* CGIs (the html code, xml code and web config tools for the Management Console) are proprietary to Opengear, however the code will be provided to customers, under NDA.

The *console server* also supports GNU *bash* shell script enabling the *Administrator* to run custom scripts. GNU *bash*, version 2.05.0(1)-release (arm-OpenGear-linux-gnu) offers the following shell commands:

```
alias [-p] [name[=value] ... ]
bg [job_spec]
bind [-lpvsPVS] [-m keymap] [-f fi break [n]
builtin [shell-builtin [arg ...]]
case WORD in [PATTERN [| PATTERN]
cd [-PL] [dir]
command [-pVv]
command [arg ...]
compgen [-abcdefjkvu] [-o option]
complete [-abcdefjkvu] [-pr] [-o o]
continue [n]
declare [-afFrxi] [-p] name[=value]
dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [jobspec ...]
echo [-neE] [arg ...]
enable [-pnds] [-a] [-f filename]
eval [arg ...]
exec [-cl] [-a name] file [redirec]
expect [arg ...]
exit [n]
export [-nf] [name ...] or export
false
local name[=value] ...
logout
popd [+N | -N] [-n]
printf format [arguments]
pushd [dir | +N | -N] [-n]
pwd [-PL]
read [-ers] [-t timeout] [-p promp]
readonly [-anf] [name ...] or read return
[n]
select NAME [in WORDS ... ;] do
COMMANDS
set [--abefhkmnptuvxBCHP] [-o opti]
shift [n]
shopt [-pqsu] [-o long-option] opt
source filename
suspend [-f]
test [expr]
time [-p] PIPELINE
times
trap [arg] [signal_spec ...]
true
type [-apt] name [name ...]
```

fc [-e ename] [-nlr] [first] [last]
fg [job_spec]
for NAME [in WORDS ... ;] do COMMA
function NAME { COMMANDS ; } or NA
getopts optstring name [arg]
hash [-r] [-p pathname] [name ...]
help [-s] [pattern ...]
history [-c] [-d offset] [n] or hi
if COMMANDS; then COMMANDS; [elif jobs [-lnprs] [jobspec ...] or *job kill [-s sigspec | -n signum | -si let arg [arg ...]*
typeset [-afFrxi] [-p] name[=value
ulimit [-SHacdflmnpstuv] [limit]
umask [-p] [-S] [mode]
unalias [-a] [name ...]
unset [-f] [-v] [name ...]
until COMMANDS; do COMMANDS; done
variables - Some variable names an
wait [n]
while COMMANDS; do COMMANDS; done { COMMANDS ; }