APPENDIX A

CLI Utilities

LMS provides Command Line Interface (CLI) support. The CLI utilities that are supported by LMS are:

- CWCLI
- Performance Tuning Tool
- syslogConf.pl Utility
- Software Management CLI Utility

CWCLI

CiscoWorks Command Line Framework (CWCLI) is the interface or framework through which application functionality is provided.

The following are the `cwcli` applications:

- `cwcli config` is the configuration command-line tool. `cwcli netconfig` command lets you use NetConfig from the command line.
- `cwcli export` is a command line tool that also provides servlet access to inventory, configuration and change audit data.

  This can be used for generating inventory, configuration archive, and change audit data for devices in LMS.
- `cwcli inventory` is a Device Management application command line tool. This tool can be used for checking the device credentials, exporting the device credentials. You can also view the devices and delete the devices.
- `cwcli invreport` is a CiscoWorks command line tool which allows you to run previously created Inventory Custom Reports and also system reports. The output is displayed in the Comma Separated Value (CSV) format.
- `cwcli netshow` is a comand line tool that lets you use NetShow features from the command line. You can use the cwcli netshow commands to view, browse, create, delete, and cancel NetShow jobs and Command Sets.

This appendix contains the following sections:

- Overview: CLI Framework (cwcli)
- Overview: cwcli config Command
- Overview: cwcli netconfig Command
- Overview: cwcli export Command

- Overview: cwcli inventory Command
- Overview: cwcli invreport Command
- Overview: cwcli netshow Command

You can set the debug mode for CLIFramework and ConfigCLI in the Log Level Settings dialog box (**Admin > System Preferences > Loglevel Settings**).

Overview: CLI Framework (cwcli)

CLI Framework (`cwcli`) is a Command-Line Interface. This interface provides application-related functionality.

The CLI Framework supports the following tasks:

- Parsing the command line for the applications.
- Easy logging and messaging capabilities
- Authentication and authorization for individual applications
- Remote access support.

This section contains:

- cwcli Global Arguments
- Remote Access

SYNOPSIS

The command line syntax is as follows:

`cwcli` *application command GlobalArgs AppSpecificArguments*

- *application* specifies one or more LMS applications that use the framework. For example, config, export, inventory, invreport, and netconfig.
- *command* specifies which core operations are to be performed for a particular service.
- *GlobalArgs* specifies arguments common for all CLI. For example, username, password, log, debug, etc.
- *AppSpecificArguments* are the additional parameters required for each core command.

You should enter the application name immediately after `cwcli` and the command name, after the application name. All other GlobalArgs arguments can be specified in any order.

Apart from the applications, Global args (`-u` *user*, `-p` *password*, `-l` *logfile*, `-m` *email*, `-d` *debuglevel*) framework also supports two generic commands. They are:

- `-v`—Version of the CLI interface.
- `-help`—All the applications that can be invoked using the framework.

SYNTAX

```
cwcli  -v
cwcli  -help
```

cwcli Global Arguments

The following table shows the **cwcli config** command arguments you can specify with all commands.

cwcli arguments	Description
-u *userid*	User ID. Field is required.

-p *password*	It is the password for the specified User ID.
	If you enter the password at the command line, a message appears:
	`* Warning * The -p option is highly insecure and *not* recommended. See -u option for more details.`
	If the password is not specified in the command line, framework searches for the password in the file pointed to by the CWCLIFILE environment variable. If the variable is not set, you are prompted to enter the password.
	`* Warning * CWCLIFILE Environment variable not set. Enter your password`
	See Setting CWCLIFILE Environment Variable for more details.

-device *devicename* or *device_list*	Device Name of the device added into DCR. You can use comma separated displaynames and wild card character **%**. For example, if there are two devices with names Rtr12 and Rtr13, Rtr% will display both the devices. To use all the devices, use **-device** %.
-view *view_list*	If the data needs to be generated for all the devices in a specific group, you can use the **-view** argument. You can use this argument to generate data for devices in all device views including system-defined groups and user-defined groups. You can enter multiple group name separated using a comma. For view name, you have to enter the fully qualified path as in the Group Administration window. To separate the path you must use forward slash only. For example, -**view** "/RME@*ciscoworks_servername*/All Devices"
-ipaddress *address*	Device IP4 address as entered in the Device and Credential Repository. You can enter multiple IP address with comma separated. You cannot use this option with **-device**, **-view** , or **-input**. Also, you cannot specify wildcard characters.
-l *logfile*	Must be a relative name. By default ConfigCLI.log and cli.log files are used, located at: • *NMSROOT*\log (On Windows) • /var/adm/CSCOpx/log (On Solaris and Soft Appliance) If the relative name is specified then the log messages are logged into the file specified. The file is created under the log directory, mentioned above. For example, **cwcli config export -u** *alpha* **-p** *beta* **-device % -l** *export.log*. In this case, export.log is created under the log directory mentioned above.
-m *email*	Email address to mail the command output to. You can enter single or comma separated email IDs.
-d *debuglevel*	Enables debugging to command-line tool. Specifies debugging verbosity. Default is least verbose.

cwcli arguments	Description
`-help`	Displays usage information.
`-input`	Text file containing arguments for each device.

Note `-d` and `-l` arguments are supported for backward compatibility. Select **Admin > System > Debug Settings > Config and Image Management Debugging settings > CLI Framework** to set debug levels.

When using wildcards, you must use the percent sign (`%`), not an asterisk (`*`), as shown in the following examples:

```
%device (lists all devices that end with the suffix 'device')
dev% (lists all devices that start with the prefix 'dev')
% (lists all devices LMS manages)
```

Remote Access

CLI framework (**`cwcli`**) offers remote access facilities to allow you to invoke **`cwcli`** commands from the client in the same way as they run on the LMS server.

The name of the servlet is /rme/cwcli.

The following is the servlet to be invoked to run any command:

For post-request,

`http://`*lms-server:lms-port*`/rme/cwcli`* *payload XML file*

For get request,

`http://`*lms-server:lms-port*`/rme/cwcli?command=cwcli config`* *commandname* `-u` *user* `-p` *BAse64 encoded pwd -args1 arg1value...*

Note Use <arg> and <argval> tags when the argument is a file.

The contents of the payload xml file is as follows.

```
<payload>
   <command>
    cwcli config export -u admin -p <Base64Enoced pwd> -device 1.1.1.1 -xml
   </command>
   <arg>
   </arg>
   <arg-val>
   </arg-val>
</payload>
```

For example to run the **`cwcli config import`** comand payload.xml is as follows:

```
<payload>
   <command>
     cwcli config import -u admin -p <Base64Enoced pwd> -device 10.77.240.106
     <arg>
```

```
      -f
      </arg>
      <arg-val>
       tempfile
      </arg-val>
    </command>
</payload>
```

The Remote Access servlet creates a temporary file with the contents specified between the arg-val tags for the **import** command. On the server the command is run as
cwcli config import -u *admin* **-p** *Base64Enoced pwd* **-device** *10.77.240.106* **-f** *tempfile*

Here, the tempfile contains the configuration of the device that you want to import.

For example,

perl samplescript.pl http(s)://*lms-server:lms-port***/rme/cwcli** *payloadXML*

To invoke the servlet using a script, see the Sample Script to Invoke the Servlet.

The script and the payload file should be residing in the client machine.

Note For the secure mode (HTTPS) the port number is 443. The default port for LMS server in HTTP mode is 1741.

Sample Script to Invoke the Servlet

```
#!/opt/CSCOpx/bin/perl
use LWP::UserAgent;
$temp = $ARGV[0] ;
$fname = $ARGV[1] ;
print "\n argv[0} = $ARGV[0] , fname = $fname \n";
open (FILE,"$fname") || die "File open Failed $!";
while ( <FILE> )
{
$str .= $_ ;
}
#print $str ;
url_call($temp);
#-- Activate a CGI:
sub url_call
{
my ($url) = @_;
my $ua = new LWP::UserAgent;
$ua->timeout(1000);
# you can set timeout value depending on number of devices
my $hdr = new HTTP::Headers 'Content-Type' => 'text/html';
my $req = new HTTP::Request ('GET', $url, $hdr);
$req->content($str);
print "It comes here \n";
my $res = $ua->request ($req);
```

```
my $result;
#    print "It comes here too \n";
if ($res->is_error)
{
print "ERROR : ", $res->code, " : ", $res->message, "\n";
$result = '';
if ($res->message =~ /read timeout/)
{
print "ERROR:Timeout has occured. Increase the timeout value in samplescript.pl.\nFor
example, if the devices managed in network is more than 1K, increase the timeout value
to 5000.";
}
}
else {
    $result = $res->content;
       if($result =~ /Authorization error/)
{           print "Authorization error\n";
          }
else {
print "\n $result" ;
         }
     }
}
```

Setting CWCLIFILE Environment Variable

You can store your username and password in a file and set a variable *CWCLIFILE* that points to the file. This helps you to avoid the **-p** argument, which will reveal the password in clear text in CLI.

You should maintain this file and control access permissions to prevent unauthorized access.

If *CWCLIFILE* is set only to filename instead of full path, **cwcli** framework looks for the current working directory.

If you use the **-p** argument, even after setting the *CWCLIFILE* variable, the password is taken from the command line instead of *CWCLIFILE*. This is not secure and usage of this argument is not recommended.

The password must be provided in the file in the following format:

username password

where username and password are the LMS login credentials. The delimiter between the username and password is a single space.

You must enter comma as the delimiter if the password is blank. Otherwise, `cwcli` framework will fail to validate the password.

Example to run the cwcli command with the `CWCLIFILE` file:

On Windows, at the command prompt enter:

`C:\Program Files\CSCOpx\bin>`**`set CWCLIFILE=`***`D:\ciscoworks\password.txt`*

`C:\Program Files\CSCOpx\bin>`**`cwcli export changeaudit -u`** *admin* **`-view`** "**`/RME@`***`ciscoworksservername/Normal Devices`*"

Where the file, password.txt contains the username and password for LMS server.

Overview: cwcli config Command

The `cwcli config` command-line tool performs the following core functions on one or more devices and the configuration archive:

- Moves configuration files from the configuration archive to one or more devices.
- Transfers the configuration files from devices to the archive if the configuration running on a device is different from the latest archived version
- Imports configuration files from the file system and pushes them to one or more devices, which updates the configuration archive
- Merges the startup configuration files with the running configuration files
- Copies the running configuration files to the startup configuration files
- Copies a configuration file to the startup configuration files
- Copies the difference between a configuration file and the running configuration to the running configuration files. This makes the configuration in the file available on the running configuration.
- Reboots running devices to load a running configuration with its startup configuration

In addition, `cwcli config` performs the following core functions on the configuration archive:

- Exports configurations from the archive to the filesystem
- Compares any two configuration files in the archive based on version or date
- Deletes configurations older than a specified date from the configuration archive

This section contains:

Using the cwcli config Command for Batch Processing

In addition to using the graphical-based device configuration functions, you can use the `cwcli config` command-line utility to perform batch processing tasks on the configuration archive, devices, or on both.

For more details see these sections:

- cwcli config Core Arguments
- Examples of cwcli config

On platforms other than Windows 2000, all files created by `cwcli config` are owned by casuser. They belong to the same group as the user (casuser) who created the files, and have read-write access for both casuser and the group.

Note Your login determines whether you can use this argument.

Getting Started With cwcli config

`cwcli config` is a command-line tool. This tool is like an interface between the user and the device and the configuration archive.

Generally, the configuration archive automatically registers modifications to the device's configuration in archived, version-based files. Over time, multiple configurations of a device accumulate in the archive. Typically, the latest version is the configuration running on the device.

Uses of cwcli config

With `cwcli config`, you can:

- Device and Archive Updates

  Modify a device's running configuration. You can allow personnel of your organization to modify the device's configuration without explicitly providing them with Telnet access to the device.

- Deleting Configurations

  Delete unwanted versions of the configuration file from the archive. This is a command-line variant of the UI purge feature.

- Comparing Configurations

  Generate 'diffs' of different configuration versions of the same device to find out what modifications were made. This is a command-line counterpart for GUI-based reports.

Device and Archive Updates

Whenever you use `cwcli config` to update the running configuration of the device, the tool also archives the newly written configuration to the archive, bypassing the auto-detection mechanism.

Getting a Version of the Device Configuration

To obtain a version of the device's configuration from the device, modify it, and then write it back to the device. You use two features of `cwcli config` to do this.

1. Use the `export` command to obtain a copy of the desired configuration version file.
2. Edit and deploy it on the device using the `import` function. If the update succeeds, `import` also archives the configuration in the archive as the latest version.

Example:

`cwcli config export -u` *user* `-p` *pass* `-device` *zebra.domain.com* `-version` *3* `-f` *zebraconf*

version 3 of device zebra's configuration has been obtained from the device. It is available in the file zebraconf. You must edit the file and make the necessary modifications.

```
cwcli config import -u user -p pass -device zebra.domain.com -f zebraconf
```

The edited file is written back to the device and archive. If there were five configurations originally, a sixth one is now added.

If you want to update the running config on the device, and are certain that the latest archived version is the same as the running config, then you can obtain the latest version as follows:

cwcli config export -u *user* **-p** *pass* **-device** *zebra.domain.com* **-f** *zebraconf*

the latest version is copied to file zebraconf.

After writing the edited configuration to the device, you might want to reboot the device. You can do this automatically from **cwcli config** by using the **-reboot** argument to the **import** command:

cwcli config export -u *user* **-p** *pass* **-device** *zebra.domain.com* **-f** *zebraconf* **-reboot**

In addition, you might want to write file zebraconf to both the running as well as the startup configuration. To do this, enter the following command:

cwcli config export -u *user* **-p** *pass* **-device** *zebra.domain.com* **-f** *zebraconf* **-save**

Reverting to Earlier Configuration Version

For running configuration, use either compare or export to decide, which version to revert to.

For VLAN configuration, look into the Configuration Version Report for the device to find the versions for which VLAN configuration is also archived. Then use **put** to deploy the desired version.

The **put** function gets the requested version from the archive, writes it to the device. For Running configuration, it archives it as the latest version of that device.

Example:

cwcli config put -u *user* **-p** *pass* **-device** *zebra.domain.com* **-version** *3*

version 3 of device zebra's configuration is extracted from the archive and written to the device. It is also stored in the archive as the latest version.

Example:

cwcli config put -u *user* **-p** *pass* **-device** *zebra.domain.com* **-version** *3* **-filetype** *vlan*

version 3 of device zebra's vlan configuration is extracted from the archive and written to the device.

Like **import**, the **put** function allows you to reboot the device using the **-reboot** argument, and to update the startup configuration using the **-save** argument.

Writing Startup Configuration to Running Configuration

To write the startup configuration of the device to its running configuration. Use the **start2run** function of **cwcli config** to retrieve the startup configuration from the device, and then write it back to the device's running configuration. The new running configuration is archived as the latest version.

Example:

cwcli config start2run -u *user* **-p** *pass* **-device** *zebra.domain.com*

To ensure that the running configuration on the device is stored in the archive, that is, synchronize the archive with the device. Use the get function to do so.

Example:

cwcli config get -u *admin* **-p** *admin* **-device** *zebra.domain.com*

The running configuration of device zebra is retrieved from the device and archived as the latest version, only if there is a need to do so. However, if the running configuration does not differ from the latest archived version, then the archival does not take place.

Configuration updates can be performed on multiple devices at once. For more details see "Running cwcli config on Multiple Devices" section on page A-11.

Deleting Configurations

Use the `delete` function of `cwcli config` to delete unwanted versions from the archive, to conserve disk space, and to reduce visual clutter on reports.

Example:

`cwcli config delete -u` *user* `-p` *pass* `-device` *zebra.domain.com* `-version` *2 5*

All versions between and including 2 and 5 are removed from the archive. There is also a time-stamp based variant.

Comparing Configurations

Use the `compare` function to compare any two versions of the archived configuration files of one or more devices. The compare function also lists down the entire configuration changes based on the timestamp.

Example:

`cwcli config compare -u` *user* `-p` *pass* `-device` *zebra.domain.com* `-version` 2 5

`cwcli config` can only compare the archived configuration files. The compliance report is stored in the job directories.

Remote Access

`cwcli config` uses remote access facilities offered by the CLI framework to allow you to invoke the `cwcli config` commands from the client in the same manner they would run them on the LMS server.

The name of the servlet is /rme/cwcli.

All the command can be run remotely.

Note For the secure mode (HTTPS) the port number is 443. The default port for LMS server in HTTP mode is 1741.

Running cwcli config

The `cwcli config` command is located in the following directories, where install_dir is the directory in which LMS is installed:

- On Solaris and Soft Appliance systems, *NMSROOT*/bin. The default directory is /opt/CSCOpx
- On Windows systems, *NMSROOT*\bin

  The default install directory is C:\Program Files.

If you install LMS on Windows on an NTFS partition, only users in the administrator or casuser group can access `cwcli config`.

Users with read-write access to the CSCOpx\files\archive directory and the directories under that can also use `cwcli config`.

Running cwcli config on Multiple Devices

You can run **cwcli config** simultaneously on multiple devices. Details vary from command to command. This section describes how to apply import on multiple devices. Details of multiple-device syntax for other commands are described under the DESCRIPTION in the man page.

The commands, such as **put**, **import**, **write2run** and **write2start** accept only one device on the command line. If you want to apply the command to multiple devices, enter the names of those devices and any arguments in a text file.

For example, assume that you want to deliver the configuration file serviceconf to devices, antelope and rhino. Also assume that you want to reboot rhino. The command line of **cwcli config** is as follows:

cwcli config import -u *admin* **-p** *admin* **-input** *device-list* **-m** *root@netcontrol.domain.com*

You do not want the output of the command to go to stdout. Instead, you want it to be mailed to the superuser at host netcontrol.

Device-list is a text with the following contents:

```
# comments start with a leading hash symbol. Write serviceconf to rhino and # antelope.
reboot antelope.
-device rhino.domain.com -f serviceconf
-device antelope.domain.com -f serviceconf -reboot
# end of input file device-list
```

Additional Information

The examples in this man page are not comprehensive. There are many other scenarios in which **cwcli config** can be used.

For example, if you want to modify the running configuration on the device, without using the latest archived version, considering the latest may not be the same as the running configuration. You can apply the get command and then export and import. Various combinations of the features can be used.

You can also use **cwcli config** in UNIX cron jobs to schedule config updates in advance.

Also, the output generated by **cwcli config** can be logged to a file and sent to any recipient through email. A host of additional arguments can be applied on other commands.

cwcli config Command Parameters

Using the **cwcli config** commands you can manipulate, deploy and archive your device configuration files.

Using the Compare Command

When you specify the **compare** command, both **-version** and **-date** are optional.

- If you do not specify **-version** or **-date**, the latest configuration is compared with the previous version.

- If you do specify `-version` or `-date`, and the value you enter is the latest version or date, that configuration is compared with the previous version.

Using the Delete Command

When you specify the `-date` command, you must specify `-version` or `-date`.

If you specify only one date, all versions archived up and including that date are deleted.

To delete a version archived on a particular date, specify two dates that are the same date as the archived version date. The latest two versions of configuration can never be deleted from the archive. Be careful while using the `delete` command.

Parameters For All cwcli config Commands

The `-d` and `-l` arguments are supported for backward compatibility.

In LMS, select **Admin > System > Debug Settings > Config and Image Management Debugging settings > ConfigCLI** to set debug levels.

When using wildcards, you must use the percent sign (%), not an asterisk (*), as shown in the following examples:

```
%device
dev%
%device%
```

The following table lists the `cwcli config` command-specific arguments and which commands you can use the arguments with:

cwcli config arguments	Applicable Commands	Description
`-baseline`	`createdeployparamfile`, `directbaselinedeploy`	Specifies the name of the Baseline template for which the parameter file has to be created.
`-date`	`compare`, `delete`	• Compare – If you specify one date, the latest configuration version is compared with the most recently archived version on that particular date. – If you specify two dates, the most recently archived version of the first date is compared with the most recently archived version of the second date. • Delete – If you specify one date, all versions archived up to this date are deleted. – If you specify two dates, all versions archived between and on those dates are deleted.
`-enable_pass`	`import`, `put`, `write2run`, `write2start`, `run2start`, `start2run`, `deploycomplianceresults`, `compareanddeploy`, `reload`	Specifies execution mode Base64 encoded Password for connecting to device.

cwcli config arguments	Applicable Commands	Description
`-f` *filename*	`export`, `import`	Specifies fully qualified pathname of configuration file to import to or export from. • If you do not specify this argument, the current working directory is assumed. • If you do not specify this argument when importing or exporting a single device configuration, default filename, devicename.cfg, in the current working directory is assumed. The `-f` argument applies only to single devices. To perform the operation on multiple devices, you must specify the `-input` argument.

`-input` *inputlist*	Applicable to all commands except `compareanddeploy`, `createdeployparamfile`, `deploycomplianceresults`, and `directbaselinedeploy`,	You must enter `-input` inputlist to run commands, such as put and import, on multiple devices.
		The parameter, inputlist is a text file containing arguments for each device. A line starting with # is treated as a comment.
		For example, an input list file might look like this:
		`#comment line`
		`-version version [-save] [-reboot] device_name`
		`-version version [-save] [-reboot] device_name`

`-jobid`	`createdeployparamfile`	Used to specify the job identifier of the previously run comparewithbaseline job.
`-l`	`createdeployparamfile`, `directbaselinedeploy`	Specifies the file to log the results of the command.
`-listonly`	`write2run`	Displays difference between the latest running configuration for device in configuration archive and new configuration that is generated, without downloading changes.
`-m`	`createdeployparamfile`, `directbaselinedeploy`	Specifies an email address to send the results of the command.
`primary_pass`	`import`, `put`, `write2run`, `write2start`, `run2start`, `start2run`, `deploycomplianceresults`, `compareanddeploy`, `reload`	Specifies primary user name for connecting to device.
`-primary_user`	`import`, `put`, `write2run`, `write2start`, `run2start`, `start2run`, `deploycomplianceresults`, `compareanddeploy`, `reload`	Specifies primary user name for connecting to device.
`-reboot`	`import`, `put`	After successfully pushing a configuration to a device, device reboots. By default the device does not reboot. For IOS devices, you must also specify -save to avoid losing configuration changes when rebooting.
`-save`	`import`, `put`	Applies to Cisco IOS devices only. Performs a write memory after pushing the configuration. The default is no write memory.

cwcli config arguments	Applicable Commands	Description
`-timeout`	**import**, **put**, **write2run**, **write2start**, **run2start**, **start2run**, **comparewithbaseline**, **deploycomplianceresults**, **compareanddeploy**, **get**, **reload**	Specifies the duration of the interval in seconds between two successive polling cycles. Configuration Archive is polled according to the interval specified to retrieve and display the job results.
`-version` *version*	**compare**, **delete**, **export**, **put**	• For put and **export**, you can specify one version of the configuration in the archive. • For compare, you can specify two versions, which are compared with each other. If you specify only one version, that is compared with latest archived version. • For delete, if you specify one version, that version is deleted. If you specify two versions, all versions in between and including those version are deleted.

cwcli config Syntax Examples

The following examples demonstrate the **cwcli config** command syntax. Square brackets ([ ]) indicate arguments. A pipe (|) acts as a delimiter. This means that only one of the listed entries can be specified.

Note Make sure you first use the **cwcli config** command in a test environment before running the command in production. This is to avoid any loss of data when a device is rebooted or a configuration is overwritten.

The following command extracts the running configurations from all devices:

```
cwcli config get -u user -p password -device %
```

The following command exports the configuration of all the devices from the archive and puts the configuration into the file, devicename.cfg. This is the default file name because **-f** is not specified:

```
cwcli config export -u user -p password -device %
```

If there is more than one device in the default view All, you see an error message because the **export** command does not accept multiple device names on the command line. You must specify the **-input** argument to run the **export** command on more than one device.

The following table shows more syntax examples:

Argument	Syntax	Notes
no arguments	**cwcli config -u** *user* **-p** *password* [**-v -help**]	If you do not specify arguments, **cwcli config** shows command usage (**-help**)
compare	**cwcli config compare -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* |**-ipaddress** *list* } { **-version** *version1* [*version2*] | **-date** *date1* [*date2*] }	Specify versions to compare using **-version** or **-date** argument. When specifying a date, use format mm/dd/yyyy. If you do not specify a date or a version, the latest two archived configurations are compared.
compareanddeploy	**cwcli config compareanddeploy -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] {**-device** *list* | **-view** *name* | **-device** *list* **-view** *name* |**-ipaddress** *list* }{ **-baseline** *baselinefile* }[**-timeout** *seconds*] [**-input** *argumentFile*] [**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*]	Creates a job that compares the given Baseline template with the latest version of the configuration for a device and downloads the configuration to the device if there is non-compliance.
comparewithbaseline	**cwcli config comparewithbaseline -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* |**-ipaddress** *list* }{ **-baseline** *baselinefile* }[**-timeout** *seconds*] [**-input** *argumentFile*]	Creates a job that compares the given Baseline template with the latest version of the configuration for a device. In case of non-compliance, the non-compliant commands are displayed.
delete	**cwcli config delete -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* |**-ipaddress** *list* } { **-version** *version1* [*version2*] | **-date** *date1* [*date2*] }	Deletes the specified device configuration from the archive. Use **-date** or **-version** argument to specify configurations to delete. If you specify two dates, all configurations archived between those dates are deleted. If you specify two versions, all configurations between and including the versions are deleted.
deploycomplianceresults	**cwcli config deploycomplianceresults -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-substitute** *datafile* } {**-jobid** *jobID*}[**-timeout** *seconds*][**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*]	Creates a job that uses the previously executed **comparewithbaseline** job to get the non-compliance commands and create a job. It replaces the parameters in the non-compliant commands with the values from the data file. The commands are then downloaded to ensure compliance with the baseline configuration.
export	**cwcli config export -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *displayName* **-view** *name* | **-ipaddress** *list* } [**-f** *filename*] **[-version** *number*] [**-xml**] [**-input** *argumentFile*]	Retrieves a configuration version for a device from the archive and writes it to a file. Exported configurations are named devicename.cfg if **-f** argument is not used.

Argument	Syntax	Notes
`get`	`cwcli config get -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*][`-timeout` *seconds*] [`-filetype` *running|startup|runningstartup*] { `-device` *list* | `-view` *name* | `-device` *list* `-view` *name* |`-ipaddress` *list* }	Creates a job that fetches the configuration from the device and stores it in the archive.
`import`	`cwcli config import -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*][`-timeout` *seconds*] { `-device` *displayName* |`-ipaddress` *address* } [`-f` *filename*] [`-save` [`-reboot`]][`-input` *argumentFile*]	Creates a job that retrieves the configuration from a file and transfers it to the device. The job is added to the device running configuration. It then polls Configuration Archive at periodic intervals to get the job results and display it. Specify `-input` to operate on more than one device. You cannot specify wildcards or more than one device.
`listversions`	`cwcli config listversions -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] { `-device` *list* | `-view` *name* | `-device` *displayName* `-viewname` | `-ipaddress` *list*} `-baseline`	Lists the versions of the configuration archived for a device on the main branch or the Baseline templates applicable to a device.
`put`	`cwcli config put -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] { `-device` *displayName* |`-ipaddress` address `-version` *number*}[`-config` *1|2*][`-save` [`-reboot`]] [`-input` *argumentFile*][`-timeout` *seconds*] [`-filetype` *vlan|running*][`-primary_user` *primary user name*] [`-primary_pass` *Base64 encoded primary password*] [`-enable_pass` *Base64 encoded enable password*]]	Creates a job that retrieves the configuration from the configuration archive and pushes it to the device. Specify `-input` to operate against more than one device. You cannot specify wildcards or more than one device. You must specify a version.
`reload`	`cwcli config reload -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*][`-l` *logfile*] { `-device` *list* | `-view` *name* | `-device` *list* `-view` *name* |`-ipaddress` *list* }[`-input` *argumentFile*][`-timeout` *seconds*][`-primary_user` *primary user name*] [`-primary_pass` *Base64 encoded primary password*] [`-enable_pass` *Base64 encoded enable password*]	Creates a job that reboots devices. The configuration loaded runs with the startup configuration.
`run2start`	`cwcli config run2start -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*][`-l` *logfile*]{ `-device` list | `-view` *name* | `-device` *list* `-view` *name* | `-ipaddress` *list*}[`-config` 1|2] [`-input` *argumentFile*][`-timeout` *seconds*][`-primary_user` *primary user name*] [`-primary_pass` *Base64 encoded primary password*] [`-enable_pass` *Base64 encoded enable password*]	Creates a job that overwrites the startup configuration of device with running configuration. Specify multiple devices with `-device` argument by separating each device name with comma or with `-input` argument, which takes filename containing the multiple devices as an argument.

Argument	Syntax	Notes
`start2run`	`cwcli config start2run -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*][`-l` *logfile*] { `-device` *list* | `-view` *name* | `-device` *list* `-view` *name* | `-ipaddress` *list* } [`-config` 1|2] [`-input` *argumentFile*][`-timeout` *seconds*] [`-primary_user` *primary user name*] [`-primary_pass` *Base64 encoded primary password*] [`-enable_pass` *Base64 encoded enable password*]	Creates a job that merges the startup configuration with running configuration. Specify multiple devices with `-device` argument by separating each device name with comma or with `-input` argument, which takes filename containing the multiple devices as an argument.
`write2run`	`cwcli config write2run -u` *userid* `-p` *password* [`-d` *debuglevel*][`-m` *email*][`-l` *logfile*] { `-device` *displayName* | `-ipaddress` *address* } `-f` *filename* [`-config` *1|2*][`-listonly`][`-input` *argumentFile*][`-timeout` *seconds*][`-primary_user` *primary user name*][`-primary_pass` *Base64 encoded primary password*][`-enable_pass` *Base64 encoded enable password*]	Creates a job that downloads the differences between the specified file and the latest version in the archive for the specified device. If you specify `-listonly`, difference is displayed but no changes are downloaded. To run command on multiple devices, use `-input` argument, which takes a filename as an argument.
`write2start`	`cwcli config write2start -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*][`-l` *logfile*] { `-device` *displayName* | `-ipaddress` *address* `-f` *filename*} [`-config` 1|2][`-input` *argumentFile*][`-timeout` *seconds*][`-primary_user` *primary user name*][`-primary_pass` *Base64 encoded primary password*] [`-enable_pass` *Base64 encoded enable password*]	Creates a job that erases contents of device startup configuration and writes contents of given file as new startup configuration. You must specify a filename. To run a command against multiple devices, use `-input` argument.

Argument	Syntax	Notes
`collectiondate`	`cwcli config collectiondate -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] [`-filetype` *running*|*startup*|*vlan*] [`-input` *argumentFile*] { `-device` *list* `-view` *name* |`-ipaddress` *list* }	Displays the last config collection date for the devices. You can specify a filename by using the `-input` argument. The input file should be of this format: `-device` 1.1.1.1,2.2.2.2,3.3.3.3 `-filetype` *vlan* or `-filetype` *vlan* `-device` 1.1.1.1,2.2.2.2,3.3.3.3 The `-filetype` should be either vlan, running or startup. If `-filetype` is not specified, then running will be taken as the default filetype value. The output contains device name, time of last config collection, and the filetype separated by comma.
`accessdate`	`cwcli config accessdate -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] [`-filetype` *running*|*startup*|*vlan*] [`-input` *argumentFile*] { `-device` *list* `-view` *name* |`-ipaddress` *list* }	Displays the last config collection attempt date for the devices. You can specify a filename by using the `-input` argument. The input file should be of this format: `-device` 1.1.1.1,2.2.2.2,3.3.3.3 `-filetype` *vlan* or `-filetype` *vlan* `-device` 1.1.1.1,2.2.2.2,3.3.3.3 The `-filetype` should be either vlan, running or startup. If `-filetype` is not specified, then running will be taken as the default filetype value. The output contains device name, time of last attempt, and the filetype separated by comma.

cwcli config Core Arguments

cwcli config Argument	Description
`compare`	Compares last two configurations in archive, specific configuration versions, or configuration changes based on a specified date. To run this command on multiple devices, specify `-device` argument or `-input` argument.
`delete`	Deletes configurations older than specified date or version from archive. To run this command on multiple devices, specify `-device` argument or `-input` argument.
`export`	Retrieves latest configuration from archive and writes it to specified file. To run this command on multiple devices, specify `-input` argument.

cwcli config Argument	Description
`get`	Pulls configuration from device to configuration archive if configuration is different from latest archived configuration. To run this command on multiple devices, specify `-device` argument or `-input` argument.
`import`	Imports configuration from specified file and pushes it to devices. To run this command on multiple devices, specify `-input` argument.
`put`	Pushes configuration files from the configuration archive to device based on version. To run this command on multiple devices, specify `-input` argument.
`reload`	Reboots devices to reload running configuration with startup configuration. To run this command on multiple devices, specify `-device` argument or `-input` argument.
`run2start`	Overwrites startup configuration with running configuration. To run this command on multiple devices, specify `-device` argument or `-input` argument.
`start2run`	Merges startup configuration with running configuration. To run this command on multiple devices, specify `-device` argument or `-input` argument.
`write2run`	Downloads difference between latest running configuration for the device in configuration archive with configuration in file specified by -f argument. To run this command on multiple devices, specify `-input` argument.
`write2start`	Erases the contents of the device's startup configuration and writes the contents of the given file as the device's new startup configuration. To run this command on multiple devices, specify `-input` argument.
`collectiondate`	Displays the last config collection date for the devices. To run this command on multiple devices, specify `-device` argument or `-input` argument.
`accessdate`	Displays the last config collection attempt date for the devices. To run this command on multiple devices, specify `-device` argument or `-input` argument.

Examples of cwcli config

The following `cwcli config` command retrieves configurations for all devices in the LMS home_routers domain and stores the configurations in Sybase:

```
cwcli config get -u adam -p max -view home_routers
```

where home_routers is a device view.

The following `cwcli config` command reads inputfile and, for each device listed, pushes the appropriate configuration to that device:

```
cwcli config import -U adam -P max -input /tmp/inputfile
```

cwcli config Command Man Page

This man page is also accessible from the command line of a LMS server installed on a UNIX system.

To view the man page, add the path install_dir/CSCOpx/man to the MANPATH variable. Then you can enter the command man `cwcli config` from any directory.

You can also access man pages for each **cwcli config** command by entering the command man cwc-command, where command is the command name (for example, **export**).

The man pages for each subcommand are also available in this help system.

NAME

cwcli config LMS command line interface for the device configuration archive

SYNOPSIS

cwcli config *command* {**-arg1** [*arg1Value*] **-arg2** [*arg2Value*] **-argN** [*argNValue*]}

cwcli config -help

DESCRIPTION

cwcli config is a LMS command line tool that allows you to access the configuration archive or configurations on devices. You can use **cwcli config** to update, export, and import configurations on devices and in the archive. You can also compare configurations and delete old configurations.

To get a list of supported commands, run the command

cwcli config -help

or

cwcli config?

Help on each command can be obtained in the following manner:

cwcli config *command* **-help**

For example:

cwcli config export -help

Additionally, man pages are available on UNIX installations for individual commands. To view the man page for any command, enter:

man cwc-*command*

For example:

man cwc-export

Arguments

Many of the arguments are common across all commands. These arguments can be broadly classified as those that are expected by every command (function independent) and those that are specific to the context of a command.

Mandatory Arguments

You must use the following arguments with all commands.

-u *userid*

Specifies the LMS username. You must define an environment variable `cwcli CWCLIFILE` with value set to a filename, which will contain the corresponding password.

The file has to be maintained by you. You can control the access permissions of this file to prevent un-authorized access. **`cwcli config`** looks for current working directory if `cwcli CWCLIFILE` is set to only file name instead of full path.

If **-u** argument is used along with **-p** argument, the password is taken from the command line instead of `cwcli CWCLIFILE`. This is not secure and usage of this argument is not recommended.

The password must be provided in the file in the following format:

```
username password
```

Where username is the LMS user name given in command line. The delimiter between username and password is single blank space. You must provide the delimiter if the password is blank

Otherwise, **`cwcli config`** will not validate the password. The password file can contain multiple entries with different user names. The password of the first match is considered in case of duplicate entries.

See Setting CWCLIFILE Environment Variable for more details.

Function-independent Arguments

You can use the following arguments without any commands:

-help

When run with the **-help** argument, **`cwcli config`** displays a list of all supported commands and a one-line description of the command.

-v

When run with the **-v** argument, **`cwcli config`** displays **`cwcli config`** version information.

Function-dependant Arguments

You can use the following arguments only with commands:

-p *password*

Specifies the password for the LMS username.

Warning **SECURITY WARNING: If -p *password* is used, the password is read from the command line instead of `cwcli CWCLIFILE`. This is highly insecure and *not* recommended. See -u argument for more details. See Setting CWCLIFILE Environment Variable for more details.**

`-d` *debuglevel*

Sets the debug level based on which debug information is printed. debuglevel is a numeric value between 1 and 5.

`-f` *filename*

Specifies the name of the file to which the retrieved configuration is written. If not specified, devicename.cfg is assumed.

`-l` *logfile*

Logs the results of the `cwcli config` command to the specified log filename.

`-m` *mailbox*

Mails the results of the `cwcli config` command to the specified email address.

Function-specific Arguments

You can use the following arguments only with specific commands:

`-baseline`

Used with the `compareanddeploy, deploycomplianceresults, listversions, createdeployparamfile, directbaselinedeploy,` or `comparewithbaseline function`, specifies the name of the Baseline template that is compared with the latest configuration version of the device.

If there are commands in the baseline configuration file that are not compliant with the latest configuration of the device in the archive, they are downloaded to the device.

Note The Baseline template must not contain any parameters for the command to succeed.

`-date` *date1 date2*

Used with the `compare` or `delete` command, specifies the configuration date(s) to compare or delete. Use the format mm/dd/yyyy.

`-device` *name*

Used with the `export, import,` or `put` function, specifies the name of the device. You can specify a wildcard, %, in the device name to match any device(s) that have the same textual pattern.

`-device` *list*

Used with the `get, start2run, compare, compareanddeploy, comparewithbaseline, deploycomplianceresults, listversions, put, run2start, start2run, write2run` or `delete` commands

Specifies the list of device names separated by commas. You can specify a wildcard, %, in the device list to match device(s) that have the same textual pattern.

- `ipaddress` *list*

Used with the `get, start2run, compare, compareanddeploy, comparewithbaseline, deploycomplianceresults, listversions, put, run2start, start2run, write2run` or `delete` commands.

Specifies IP4 address as entered in the Device and Credential Repository. You can enter multiple IP address with comma separated.

You cannot use this option with **-device**, **-view**, or **-input**. Also, you cannot specify wildcard characters.

-filetype *type*

Used with the put function, specifies the type of the configuration (running/vlan) that should be written to the device.

-f *filename*

Used with the **directbaselinedeploy, export, import, write2run or write2start function**, specifies the name of the file to which the configuration from archive should be exported to. Used with the **import** function, specifies the name of the file that contains the configuration to import.

Note **-f** argument must not be specified when **-view** or **-device** % is used. If used, the given file will be overwritten with the configuration retrieved for other devices.

-input *listfile*

Used with the **export, import, compareanddeploy, comparewithbaseline, deploycomplianceresults** or **put** function, specifies the name of the file containing the arguments for multiple devices.

The contents of the file must be similar to those described in the Input List File Format section later in this man page.

-listonly

Used with the **write2run** function, lists the differences between the running configuration and the specified configuration file.

-reboot

Used with the **import** or **put** function, reboots the device after the configuration has been written to the device.

-save

Used with the **import** or **put** function, saves the configuration written to the device to the device's memory.

-timeout

Used with the **compareanddeploy, deploycomplianceresults, import, put, run2start, start2run, write2run** or **comparewithbaseline** function, specifies the duration of the interval in seconds between two successive polling cycles.

-version number

Used with the **export** function, specifies the configuration version to retrieve from the archive. Used with the put function, specifies the configuration version to load from the archive and push to the device.

-version *version1 version2*

Used with the **compare**, or **delete** function, specifies the configuration version(s) to compare or delete.

-view *name*

Specifies the device view where the device name specified with **-device** argument is located. If **-device** argument is not specified, performs the operation on all devices in the view. More details are described in the **-view** Argument Usage section later in this man page.

-xml

Creates an XML file with the name of the device containing the configuration retrieved.

Input List File Format

For commands that do not accept multiple device names on the command line, such as put, **import**, and **export**, you can create an input list file that contains a list of devices to perform the operation on.

The contents of the input list file are a sequence of lines. Each line specifies a device name and the arguments to apply to that device. The arguments must be specific to the function. You cannot include view names in the input list file. You must specify view names on the command line. You can include comments in the input list file by starting the each commented line with #.

Input List File Example:

For the command

cwcli config put -u *userid* **-p** *password* **-view** *myView* **-input** *~/todo_list*

An example of the input list file ~/todo_list is # Comment line.

```
-version 3 -reboot -device enm-2501.cisco.com
-version 2 -save -device enm-4500.cisco.com
```

-view Argument Usage

If both **-device** and **-view** are specified, the devices in that view and the devices specified against **-device** are considered.

For example, assume that **-view** has two devices D1 and D2 and D3 is specified against **-device**, then all the three devices D1, D2 and D3 are considered.

-view Argument Usage Examples:

Search for a device in a specified view:

cwcli config export -u *admin* **-p** *admin* **-view** *myView* **-device** *myDevice*

cwcli config Subcommand Man Pages

Each **cwcli config** command has a man page. You can access these man pages from the command line of a LMS server installed on a UNIX system.

To view the man pages, add the path:

install_dir/CSCOpx/man to the *MANPATH* variable.

Then you can enter the command
man cwc- *command*

where *command* is the command name. For example, export.

This topic contains the man pages for the following **cwcli config** subcommands:

- compare
- comparewithbaseline
- compareanddeploy
- delete
- deploycomplianceresults
- export
- get

- import
- put
- reload
- run2start
- start2run
- write2run
- write2start
- listversions
- createdeployparamfile
- directbaselinedeploy
- collectiondate
- accessdate

compare

Name	`cwcli config compare` – CiscoWorks `cwcli config compare` function
Syntax	`cwcli config compare -u` *userid* `-p password` [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] { `-device` *list* | `-view` *name* | `-device` *list* `-view` *name* | `-ipaddress` *list* } { `-version` *version1* [*version2*] | `-date` *date1* [*date2*] } `cwcli config compare -help`
Description	`compare` lists the differences between versions of a device configuration. You can specify the versions to be compared by using the `-version` argument or the `-date` argument. • If you specify the `-version` argument with only one version number, that version is compared with the latest archived configuration of the device. • If you specify the `-date` argument with only one date, the configuration version with that date is compared with the latest archived configuration. When specifying a date, use the format mm/dd/yyyy. • If you do not specify either a date or a version, the latest two archived configurations are compared. You can specify multiple devices by separating each device name with a comma. The output of the Compare function can be interpreted as follows: – Lines preceded by '+' sign signify those occurring only in the first version but not in the latter. – Lines preceded by '-' sign signify those occurring only in the latter version but not in the first. – Lines preceded by '<' and '>' connote those which are present in both files but differ from each other.

compareanddeploy

Name	**cwcli config compareanddeploy** – CiscoWorks compare and download configuration with Baseline template function.
Syntax	**cwcli config compareanddeploy -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* |**-ipaddress** *list* }{ **-baseline** *baselinefile* }[**-timeout** *seconds*] [**-input** *argumentFile*] [**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*] **cwcli config compareanddeploy -help**
Description	**compareanddeploy** creates a job that compares the given Baseline template with the latest version of the configuration for a device and downloads the configuration to the device if there is non-compliance. If you specify **-baseline** argument, the name of the Baseline template is compared with the latest configuration version of the device and later downloaded to the device if there are any commands in the baseline config file which are not compliant with the latest configuration of the device in the archive. The Baseline template must not have any parameters for the command to succeed.

comparewithbaseline

Name	**cwcli config comparewithbaseline** - CiscoWorks compare configuration with Baseline template function.
Syntax	**cwcli config comparewithbaseline -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* |**-ipaddress** *list* }{ **-baseline** *baselinefile* }[**-timeout** *seconds*] [**-input** *argumentFile*] **cwcli config comparewithbaseline -help**
Description	**comparewithbaseline** creates a job that compares the given Baseline template with the latest version of the configuration for a device. If you use the **-baseline** argument, the name of the Baseline template is compared with the latest configuration version of the device.

delete

Name	**cwcli config delete** – CiscoWorks **cwcli config delete** function
Syntax	**cwcli config delete -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* | **-ipaddress** *list* } { **-version** *version1* [*version2*] | **-date** *date1* [*date2*] } **cwcli config delete -help**
Description	**delete** deletes the specified device configuration from the archive. You can use the **-date** argument or the **-version** argument to specify which configurations to delete. • If you specify two dates, all configurations archived between those two dates are deleted. • If you specify only one date, all configurations up to and including the configuration archived on that date are deleted. • If you specify two versions, all configurations between and including the two versions are deleted. • If you specify only one version, the configuration corresponding to that version is deleted.

deploycomplianceresults

Name	**cwcli config deploycomplianceresults** - CiscoWorks deploy command with baseline function.
Syntax	**cwcli config deploycomplianceresults -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-substitute** *datafile* } {**-jobid** *jobID*}[**-timeout** *seconds*][**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*] **cwcli config deploycomplianceresults -help**
Description	**deploycomplianceresults** uses the previously run **comparewithbaseline** job to get the non-compliance commands and creates a job after replacing the parameters if any in the non-compliance commands with the values from the data file and then downloads those commands to ensure the compliance with the baseline config. If you specify the **-baseline** argument, the name of the Baseline template which will be compared with the latest configuration version of the device.

export

Name	**cwcli config export** – CiscoWorks **cwcli config**'s **export** function.
Syntax	**cwcli config export -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] { **-device** *name* | **-view** *name* | **-device** *name* **-view** *name* |**-ipaddress** *list* } [**-f** *filename*] [**-version** *number*] [**-xml**] [**-input** *argumentFile*] **cwcli config export -help**
Description	**export** retrieves the configuration specified by the **-version** argument, for the device specified by **-device** and/or **-view** argument, from the archive and writes it to the file specified by the **-f** argument. • If you do not specify a version number, the latest configuration of the device from the archive is retrieved. • If you do not specify a file name, a file named devicename.cfg is created. To run this command against multiple devices, you must specify the **-input** argument, which takes a file name as an argument. The contents of the file must be similar to those described in the Input List File Format section of the **cwcli config** man page.

get

Name	**cwcli config get** – CiscoWorks **cwcli config** get function
Syntax	**cwcli config get -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] **-filetype** *running|startup|runningstartup* **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* | **-ipaddress** *list* } **cwcli config get -help**
Description	**get** retrieves the running configuration from the device(s), specified by the **-device** and/or **-view** argument, and pushes it to the configuration archive if the running configuration is different than the latest version in the archive. For devices that support vlan configuration like CatIOS devices, the vlan configuration is also fetched and archived along with running-configuration. However, if a new version of the running configuration is not archived, the vlan configuration fetched, overwrites the previously archived vlan configuration for the latest version of running configuration in the archive. You can run the get function against multiple devices by separating each device name with a comma.

import

Name	**cwcli config import** – CiscoWorks **cwcli config import** function
Syntax	**cwcli config import -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*][**-timeout** *time*] { **-device** *name* |**-ipaddress** *address*} [**-f** *filename*] [**-save** [**-reboot**]][**-input** *argumentFile*] **cwcli config import -help**
Description	**import** retrieves the configuration from a file specified by the **-f** argument, and pushes it to the device specified by the **-device** and/or the **-view** argument, adding to the device's running configuration. • If you do not specify a file name, a file named device name.cfg is used. You can specify the **-save** and **-reboot** arguments, which operate the same as for the put argument. To run the **import** argument against more than one device, you must specify the **-input** argument, which takes a file name as an argument. The contents of the file must be similar to those described in the Input List File Format section of **cwcli config**(1). The configuration archive might be updated after you specify the **import** argument if the loaded configuration is different from the latest configuration in the archive.

put

Name	**cwcli config put** – CiscoWorks **cwcli config** put function
Syntax	**cwcli config put -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] { **-device** *name* |**-ipaddress** *address* **-version** *number*}[**-config** *1|2*][**-save** [**-reboot**]] [**-input** *argumentFile*][**-timeout** *seconds*] [**-filetype** *vlan|running*][**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*]] **cwcli config put -help**
Description	**put** retrieves the configuration specified by **-version** from the configuration archive and pushes it to the device specified by the **-device** and/or **-view** argument The **-filetype** can be used to specify the type of configuration viz running/vlan configuration. By default, the running configuration is considered • In case of running configuration, the archived running configuration is merged with the running configuration on the device unless you specify **-save**, in which case, the archived configuration is also written to the device's memory. • In case of vlan configuration, the archived vlan configuration overwrites that on the device. The vlan configuration will not come into effect until the device is rebooted. You can specify **-reboot** to reboot the device after the configuration (running/vlan) is pushed to the device. To run the put command on more than one device at a time, you must use the **-input** argument, which takes a file name as an argument. The contents of the file must be similar to those described in the Input List File Format section of **cwcli config**(1).

reload

Name	**cwcli config reload** – CiscoWorks **cwcli config reload** function
Syntax	**cwcli config reload -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name*|**-ipaddress** *list* }[**-input** *argumentFile*][**-timeout** *seconds*][**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*] **cwcli config reload -help**
Description	**reload** reboots the device(s), specified by the **-device** and/or **-view** argument, resulting in the running configuration being loaded with its startup configuration. You can specify multiple devices with the **-device** argument by separating each device name with a comma.

run2start

Name	**cwcli config run2start –** CiscoWorks **cwcli config run2start** function
Syntax	**cwcli config run2start -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*]{ **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* | **-ipaddress** *list*}[**-config** *1|2*] [**-input** *argumentFile*][**-timeout** *seconds*][**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*] **cwcli config run2start -help**
Description	**run2start** overwrites the startup configuration of any device(s), specified by the **-device** and/or **-view** argument, with its running configuration. You can specify multiple devices with the **-device** argument by separating each device name with a comma or with the **-input** argument, which takes a file name as an argument. The contents of the file must be similar to those described in the Input List File Format section of **cwcli config**(1).

start2run

Name	**cwcli config start2run –** CiscoWorks **cwcli config start2run** function
Syntax	**cwcli config start2run -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-device** *list* | **-view** *name* | **-device** *list* **-view** *name* | **-ipaddress** *list* } [**-config** *1|2*] [**-input** *argumentFile*][**-timeout** *seconds*] [**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*] **cwcli config start2run -help**
Description	**start2run** merges the running configuration of any device(s), specified by the **-device** and/or **-view** arguments, with its startup configuration to give a new running configuration. You can specify multiple devices with the **start2run** argument by separating each device name with a comma or with the **-input** argument, which takes a file name as an argument. The contents of the file must be similar to those described in the Input List File Format section of **cwcli config**(1).

write2run

Name	**cwcli config write2run –** CiscoWorks **cwcli config write2run** function
Syntax	**cwcli config write2run -u** *userid* **-p** *password* [**-d** *debuglevel*][**-m** *email*][**-l** *logfile*] { **-device** *name* | **-ipaddress** *address*} **-f** *filename* [**-config** *1|2*][**-listonly**][**-input** *argumentFile*][**-timeout** *seconds*][**-primary_user** *primary user name*][**-primary_pass** *Base64 encoded primary password*][**-enable_pass** *Base64 encoded enable password*] **cwcli config write2run -help**
Description	**write2run** compares the latest running configuration for the device in the configuration archive with the configuration in the file specified by the **-f** argument to generate a new configuration that is downloaded to the device, so that the end result is that the configuration specified in the file is available on the running configuration of the device. If **-listonly** is specified, the difference between the latest running configuration for the device in the configuration archive and the new configuration that is generated is listed on the display, but no configuration is downloaded to the device. To run this command against multiple devices, specify the **-input** argument, which takes a file name as an argument. The contents of the file must be similar to those described in the Input List File Format section of **cwcli config**(1). **CAVEAT** This command is not 100% reliable in that it may not successfully overwrite the running configuration. This is due to the dependency on the underlying Diff API, which generates the configuration difference to be downloaded to the device to make the running configuration on the device same as the one specified in the file (by the **-f** argument).

write2start

Name	**cwcli config write2start –** CiscoWorks **cwcli config write2start** function
Syntax	**cwcli config write2start -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-device** *name* **-f** *filename* |**-ipaddress** *address*} [**-config** *1|2*][**-input** *argumentFile*][**-timeout** *seconds*][**-primary_user** *primary user name*][**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*] **cwcli config write2start -help**
Description	**write2start** erases the contents of the device's startup configuration and then writes the contents of the given file as the device's new startup configuration. If you do not specify a file name, it prints an error message and exits. To run this command against multiple devices, you must specify the **-input** argument, which takes a file name as its argument. The contents of the file must be similar to those described in the Input List File Format section of **cwcli config**(1).

listversions

Name	**cwcli config listversions –** CiscoWorks **cwcli config listversions** function
Syntax	**cwcli config listversions -u** *userid* **-p password** [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*]{ **-device** *name* | **-view** *name* | **-device** *name* **-view** *name* | **-ipaddress** *list*} [**-baseline**][**-input** *argumentFile*] **cwcli config listversions -help**
Description	**Listversions** (specified by "listversions") lists the different versions of configuration files archived in the archival system. If you use the **-baseline** argument, only the names of the Baseline templates are displayed. You can choose a template and use it inline with the **comparewithbaseline** and **compareanddeploy** commands.

createdeployparamfile

Name	**cwcli config createdeployparamfile** - CiscoWorks **cwcli config createdeployparamfile** function.
Syntax	**cwcli config createdeployparamfile -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*][**-jobid** *comparewithbaseline jobid*] [**-baseline** *baselinefile*] [**-f** *parameterfile*] **cwcli config createdeployparamfile -help**
Description	**createdeployparamfile** creates a parameter file if the Baseline template containing the parameters is specified. You can use the **-jobid** argument to specify the job identifier of the previously executed **comparewithbaseline** job. You can choose a template with the **-baseline** argument and specify the name of the Baseline template for which the parameter file has to be created.

directbaselinedeploy

Name	**cwcli config directbaselinedeploy** - CiscoWorks **cwcli config directbaselinedeploy** function
Syntax	**cwcli config directbaselinedeploy -u** *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*][**-l** *logfile*] { **-baseline** *baselinefile* } {**-substitute** *parameterfile*}[**-timeout** *seconds*] [**-primary_user** *primary user name*] [**-primary_pass** *Base64 encoded primary password*] [**-enable_pass** *Base64 encoded enable password*] **cwcli config directbaselinedeploy -help**
Description	**directbaselinedeploy** creates a job that downloads the given Baseline template after retrieving the values of the parameters in the template from the given parameter file. You can use the **-timeout** argument to specify the duration of the interval in seconds between the two successive polling cycles. You can use the **-baseline** to specify the name of the Baseline template which will be compared with the latest configuration version of the device and later downloaded to the device if there are any commands in the baseline config file which are not compliant with the latest configuration of the device in the archive. You can use the **-substitute** to substitute the values from the XML parameter file for the parameters specified in the template.

collectiondate

Name	`cwcli config collectiondate` - CiscoWorks `cwcli config collectiondate` function
Syntax	`cwcli config collectiondate -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] [`-filetype` *running|startup|vlan*] [`-input` *argumentFile*] { `-device` *list* `-view` *name* |`-ipaddress` *list* } `cwcli config collectiondate -help`
Description	`collectiondate` displays the last config collection date for the devices. The output contains device name, time of last config collection, and the filetype separated by comma.

accessdate

Name	`cwcli config accessdate` - CiscoWorks `cwcli config accessdate` function
Syntax	`cwcli config accessdate -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] [`-filetype` *running|startup|vlan*] [`-input` *argumentFile*] { `-device` *list* `-view` *name* |`-ipaddress` *list* } `cwcli config accessdate -help`
Description	`accessdate` displays the last config collection attempt date for the devices. The output contains device name, time of last attempt, and the filetype separated by comma.

Overview: cwcli netconfig Command

The `cwcli netconfig` command lets you use NetConfig from the command line.

This section contains cwcli netconfig Remote Access.

Caution The `cwcli netconfig` command does not validate the command arguments you use or the configuration commands that you run using it. If you enter incorrect commands you can misconfigure or disable the devices on which the job runs.

Running the cwcli netconfig Command

To use the `cwcli netconfig` command, you must be able to run the cwcli command, and you must have permissions to use the Adhoc system-defined task. For more details see topic in the section.

The command syntax is:

`cwcli netconfig` *Sub_command Common_arguments Command_arguments*

The subcommands and arguments are described in the following sections:

- Subcommands (see Subcommands)
- Common Arguments (see Common Arguments)
- Command Arguments (see Command Arguments)

Subcommands

Subcommands specify the action the command performs. Valid values for the subcommands are:

Sub Command	Description
createjob	Creates job.
deletejob	Deletes jobs.
canceljob	Cancels jobs.
jobdetails	Lists job details.
jobresults	Lists job results.
`listjobs`	Lists jobs.
`import`	Imports user-defined tasks in XML format.
`export`	Exports user-defined tasks in XML format.
`listtasks`	Lists the NetConfig user-defined tasks.

Common Arguments

Common arguments specify parameters that apply to all subcommands. Valid values for common_arguments are:

Command Argument	Description	Usage Notes
`-u` *user*	Enter valid CiscoWorks username.	None
`-p` *password*	Enter password for username. You can also specify the password in a file. See Setting CWCLIFILE Environment Variable for more details.	None

Command Arguments

Command arguments specify parameters that apply only to specific subcommands.

The conventions followed are:

- Arguments in [ ] are optional. For optional arguments, if you do not specify a value the default value that has been set by the administrator using the NetConfig UI, will become applicable.
- Arguments in { } denote that you must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).
- Arguments suffixed with + denote that you can enter multiple values separated with spaces.
- Values that contain spaces need to be entered within “ ”. For example, the job description that you provide when you use a the `createjob` command should be entered within “ ”.

Valid values for command_arguments are described in the following table:

Sub Command	Command Argument	Description	Usage Notes
createjob Allows you to create a NetConfig job.	{**-device** *comma_separated_device_names* | **-devicefile** *devicelist_filename* | **-view** *device_view_name*}	Defines devices to be configured. *comma_separated_device_names* is list of device names. *devicelist_filename* is path to device list file. Can be full pathname or filename in the local directory. The devicelist file should be of this format: **-device** 1.1.1.1,2.2.2.2,3.3.3.3 or **-device** 1.1.1.1 **-device** 2.2.2.2 **-device** 3.3.3.3 *device_view_name* is name of the device view.	Jobs can run only one device category (IOS, Catalyst, Content Engine (CE), or Content Service Switch (CSS)). Do not enter devices of multiple categories.
	{[{**-commandfile** *commandlist_filename* **-mode** {config | enable}} [**-rollbackfile** *rollback_cmdlist_filename*]] [**-taskname :** "*User defined task name*"]}	Defines configuration commands to be used. You can specify the command file path, the command mode, the rollback file and the name of the user-defined task. Specify the user-defined task name within quotes.	*commandlist_filename* is path to command file. Can be a full pathname or filename in local directory. **-mode** specifies the command mode.{config | enable} are the command mode arguments. By default, **-mode** value is set to config. This is not valid for jobs that configure Catalyst devices. For jobs on IOS, CE, or CSS devices, config is default.
			rollback_cmdlist_filename defines the rollback configuration commands for the job. It can be a full pathname to the file or a filename in the local directory. *User defined task name* is the name that you specify for the user-defined task.
	{**-description :** *"job_description "*}	Enter the description for the job you are creating.	*"job_description"* is the description you specify, for the job that you are creating. Enter this value within quotes.

Sub Command	Command Argument	Description	Usage Notes
	[{`-schedule` : *MM/dd/yyyy:HH:mm:ss* `-schedule_type`: `once`| `weekly`| `monthly`| `lastDayOfMonth`}]	`-schedule` defines time and date job will run. If you have enabled Job Approval, and later if you create the job without using the `-schedule` argument, the job will automatically be scheduled to run after 5 minutes of the job creation time. You should approve this job within 5 minutes of creating the job. If you want to schedule the job to run at any other time, use the `-schedule` argument. If not specified, job will run immediately. `-schedule_type` defines the type of job schedule.	*MM* is month (01 to 12). *DD* is day of month (01 to 31). *YYYY* is year (Example: 2004). *HH* is hours, *mm* is minutes, and *ss* is seconds in 24-hour time. If you do not specify the schedule type, the job will be an immediate job.
	[`-policyfile` *policy_filename*]	Defines job policies using a job policy file. You can specify job policies using combination of `-policyfile` argument and other optional arguments, However, you can specify each argument only once in command.	*policy_filename* is path to job policy file. Can be a full pathname or filename in local directory.
	[`-makercomments` : "*maker comments"*]	Comments from the job creator, to the job approvers, if job approval is enabled for the job.	Enter your comments within quotes.
	[`-mkemail` : *maker email id*]	E-mail ID of the job creator, for approval notifications, if approval is enabled for the job.	None
	[`-execution`: `Sequential`|`Parallel`]	Configures the job execution property, whether the jobs should be run sequentially or in parallel.	None.
	[`-startup`]: Copy running config to startup policy	Select to cause the configuration job to write the running configuration to the startup configuration on each device after configuration changes are made successfully.	None.

Sub Command	Command Argument	Description	Usage Notes
	[`-version`] : Fail on mismatch of Config Versions.	Select to cause the job to be considered a failure when the most recent configuration version in the Configuration Archive is not the same as the configuration that was running when you created the job.	`-sync` argument should be provided if this policy is selected. This argument causes the job to archive the running configuration before making configuration changes.
	[`-email` : *Job Notification email ids*]	Specify the email addresses to which the configuration job will send status notices.	Separate multiple addresses with commas.
	[`-sync`] : Synch configuration archive before deploy	Select to cause the job to archive the running configuration before making configuration changes.	None.
	[`-failure`: "`Stop on failure`" | "`Ignore failure and continue`" | "`Rollback device and stop`" | "`Rollback device and continue`" | "`Rollback job on failure`"]	Select what the job should do if it fails to run on a device.	Ensure that you place your selected value within quotes.
	[{`-primary_user` : *Primary User name* `-primary_pass` : "*Primary password*" }]	Primary username for connecting to the device. Primary password for connecting to the device.	Enter the primary password within quotes.
	[`-enable_pass` : "*Execution mode Password*"}]	Password for running commands in the execution mode, on the device.	Enter the execution mode password within quotes.
`deletejob` This subcommand allows you to delete one or more NetConfig jobs.	`-id` *job_id+*		*job_id+* specifies the ID of the job on which to act. You can specify multiple job IDs separated by spaces or commas.
canceljob This subcommand allows you to cancel a NetConfig job from the command line.	`-id` *job_id*		*job_id* specifies ID of job on which to act.
jobdetails Allows you to view details of one or more NetConfig jobs from the command line.	[`-id` *job_id+*]	Specifies ID of job on which to act.	You can specify multiple job ID separated by spaces or commas.

Sub Command	Command Argument	Description	Usage Notes
jobresults Allows you to view results of one or more NetConfig jobs from the command line.	[**-id** *job_id+*]	Specifies ID of job on which to act.	You can specify multiple job ID separated by spaces or commas.
	[**-details**]	Specifies full details of job results to be displayed.	Not specifying details will display only the summary of job execution result.
listjobs Allows you to list all NetConfig jobs from the command line.	[**-status {A(ll)** | **R(unning)** | **C(ompleted)** | **P(ending)**}]	Specifies status of jobs to list.	If status is not specified, all registered jobs are listed.
import Allows you to import user defined task in xml format to to netconfig from the command line.	{**-taskfile** *User-defined task file* }	User-defined task filename in XML format.	
export Allows you to export one or more user defined tasks created in netconfig to xml files from the command line.	{ **-task+** *User-defined task name* }	Name of the user-defined task to be exported.	You can specify multiple tasks separated by spaces or commas.
	{**-dest** *file export location* }	Path of the destination location to which the exported user-defined task file is to be copied.	
listtask		Lists the NetConfig user-defined tasks.	

Command Examples

Example 1

The command

cwcli netconfig createjob -u *username* **-p** *password* **-devicefile** *devicefile* **-commandfile** *command.file* **-failure** *Ignore failure and Continue* **-startup**

creates a NetConfig job with the following characteristics:

- Devices mentioned in *devicefile* will be configured.
- Commands in file *command.file* will run.
- Job will continue if it fails to successfully configure a device.
- Each device's running configuration will be copied to startup as soon as the device is successfully configured.
- Job will run immediately because the **-schedule** argument is not specified.

Example 2

The command

cwcli netconfig createjob -u *username* **-p** *password* **-devicefile** *devicefile* **-commandfile** *command.file* **-policyfile** *policyfile*

creates a NetConfig job with the following characteristics:

- Devices listed in the file *devicefile* will be configured.
- Commands in the file *command.file* will run.
- The file *policyfile* contains job policy arguments that determine the job policy.

Understanding cwcli netconfig Input Files

Several types of text files are available for you to use as input for the **cwcli netconfig** command and the **-createjob** subcommand. You can also use the command list type as input for user-defined tasks.

File Type	Description	Usage Notes
Device list	Lists devices on which job will run. It lists one device on each line.	Use with **-devicefile** argument. Job can run only one device category (IOS, or Catalyst). Do not list devices of multiple categories.
Command list	Lists configuration commands that job will run; one command per line.	Use with **-commandfile** argument, or to add commands to a user-defined task.
Job policy	Lists of job policy arguments; one argument per line.	Use with **-policyfile** argument.

Examples

Device List File

```
-device device_display_name1
-device device_display_name2
-device device_display_name3
-device device_display_name4
```

Command List File

```
command1
command2
command3
command4
```

Job Policy File

This file configures the job to stop running if the job fails on a device, to write the running configuration to startup after configuration changes are made.

```
-failure Stop on Failure
-sync
```

cwcli netconfig Man Page Examples

On UNIX, you can view the complete man pages by setting the MANPATH to /opt/CSCOpx/man

The following are some examples from the NetConfig man page:

Examples

Device List File Example

For the command

cwcli netconfig createjob -u *userid* **-p** *password* **-devicefile** *c7000.dev* **-commandfile** *command.file*

-description *"cwcli netconfig job"* **-mode** config

An example of the device list file c7000.dev is

```
enm-7000-1.cisco.com
enm-7000-2.cisco.com
enm-7000-3.cisco.com
enm-7000-4.cisco.com
```

Command List File Example

For the command

cwcli netconfig createjob -u *userid* **-p** *password* **-devicelist** *c7000-1,c7000-2* **-commandfile** *command.file*

-description *"cwcli netconfig job"* **-mode** config

An example of the command file command.file is

```
snmp-server community public ro
snmp-server community private rw
```

Policy File Example

For the command

cwcli netconfig createjob -u *userid* **-p** *password* **-devicefile** *c7000.dev* **-commandfile** *command.file* **-policyfile** *policy.in*

-description *"cwcli netconfig job"* **-mode** config

An example of the policy file policy.in is

-failure *"Stop on failure"*

-sync

-execution Parallel

User-defined Task XML file Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Task name="SampleTASK">
<Template mode="1" name="iproute" parameterized="false">
<Commands>
<cli>ip route 0.0.0.1 0.0.0.0 Ethernet0/0</cli>
<cli>ip route 0.0.0.2 0.0.0.0 Ethernet0/0</cli>
<cli>ip route 0.0.0.3 0.0.0.0 Ethernet0/0</cli>
```

```
<cli>ip route 0.0.0.4 0.0.0.0 Ethernet0/0</cli>
<cli>ip route 0.0.0.5 0.0.0.0 Ethernet0/0</cli>
<cli>ip route 0.0.0.6 0.0.0.0 Ethernet0/0</cli>
</Commands>
<RollbackCommands>
<cli>no ip route 0.0.0.4 0.0.0.0 Ethernet0/0</cli>
<cli>no ip route 0.0.0.5 0.0.0.0 Ethernet0/0</cli>
</RollbackCommands>
<MDFIds>268438030,273153536,272819655</MDFIds>
</Template>
</Task>
```

cwcli netconfig Remote Access

You can also perform the **cwcli netconfig** tasks using the servlet. You will have to upload a payload XML file, which contains the **cwcli netconfig** command arguments and LMS user credentials.

You have to write your own script to invoke the servlet with a payload of this XML file and the servlet returns the output either on the console or in the specified output file, if the credentials are correct and arguments are valid.

The name of the servlet is /rme/cwcli.

The following is the servlet to be invoked to run any command:

For post request,

perl *samplepost.pl* **http://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

perl *samplepost.pl* **https://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTPS mode is 443.

The schema for creating the payload file in XML format is:

```
<payload>
<command>
cwcli inventory commandname -u user -p BAse64 encoded pwd -args1 arg1value...
</command>
</payload>
```

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

The script and the payload file should be residing in the client machine.

For get request,

http://<rme-server>:<rme-port>/rme/cwcli?command=cwcli netconfig *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

https://*lms-server:lms-port***/rme/cwcli?command=cwcli netconfig** *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTPS mode is 443.

The BAse64 encoded for “admin” is YWRtaW4=.

The URL encode for,

- Double quotes (“) is %22
- Percentage sign (%) is %25

Overview: cwcli export Command

`cwcli export` is a command line tool that also provides servlet access to inventory, configuration and change audit data.

This can be used for generating inventory, configuration archive, and change audit data for devices in LMS.

This section contains:

Note You cannot run this command for the devices that are in Conflicting or Suspended state.

This tool supports the following features:

- Generating change audit data in XML format

  The tool uses the existing Change Audit log data and generates the Change Audit log data in XML format.

  See Running cwcli export changeaudit for the usage and XML schema details

- Generating configuration data in XML format

  The tool uses existing configuration archive APIs and generates latest configuration data from the configuration archive in XML format.

  Elements in the XML file are created at the configlet level in the current configuration archive. Predefined rules that currently exist in the configuration archive are used to get the configlets data.

  See Running cwcli export config for the usage and XML schema details

- Generating inventory data in XML format

  The tool has servlet access and command line utilities that can generate inventory data for devices managed by the LMS server.

  See Running cwcli export inventory Command for the usage and XML schema details

The **cwcli export** command is located in the following directories, where *install_dir* is the directory in which LMS is installed:

- On UNIX systems, /opt/CSCOpx/bin
- On Windows systems, *install_dir*\CSCOpx\bin

  The default install directory is C:\Program Files.

If you install LMS on an NTFS partition on Windows, only users in the administrator or casuser group can access **cwcli export**. Users with read-write access to the CSCOpx\files\archive directory and the directories under that can also use **cwcli export**.

You can also perform the **cwcli export** tasks using the servlet. You will have to upload a payload XML file, which contains the **cwcli export** command arguments and LMS user credentials.

You have to write your own script to invoke the servlet with a payload of this XML file and the servlet returns the output either on the console or in the specified output file, if the credentials are correct and arguments are valid.

The name of the servlet is /rme/cwcli.

The following is the servlet to be invoked to run any command:

For post request,

perl *samplepost.pl* **http://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

perl *samplepost.pl* **https://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTPS mode is 443.

The schema for creating the payload file in XML format is:

```
<payload>
<command>
cwcli inventory commandname -u user -p BAse64 encoded pwd -args1 arg1value...
</command>
</payload>
```

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

The script and the payload file should be residing in the client machine.

For get request,

http://*lms-server:lms-port***/rme/cwcli?command=cwcli export** *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

https://*lms-server:lms-port***/rme/cwcli?command=cwcli export** *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTPS mode is 443.

The BAse64 encoded for “admin” is YWRtaW4=.

The URL encode for,

- Double quotes (“) is %22 and Percentage sign (%) is %25

Using the cwcli export Command

The command line syntax of the application is in the following format:

cwcli export *command GlobalArguments AppSpecificArguments*

- **cwcli export** is the CiscoWorks command line interface for exporting inventory/config/changeaudit details into XML format.
- *Command* specifies which core operation is to be performed.
- *GlobalArguments* are the additional parameters required for each core command.
- *AppSpecificArguments* are the optional parameters, which modify the behavior of the specific **cwcli export** core command.

The order of the arguments and arguments are not important. However, you must enter the core command immediately after **cwcli export**.

The following sections describe:

- The **cwcli export** commands (See cwcli export Commands)
- The mandatory and optional arguments (See cwcli export Global Arguments)
- The default archiving location (See Archiving cwcli export Data in XML File)

On UNIX, you can view the **cwcli export** man pages by setting the MANPATH to /opt/CSCOpx/man.

The commands to launch the **cwcli export** man pages are:

- **man cwcli-export**—To launch the **cwcli export** command man page.
- **man export-changeaudit**—To launch the **cwcli export changeaudit** command man page.
- **man export-config**—To launch the **cwcli export config** command man page.
- **man export-inventory**—To launch the **cwcli export inventory** command man page.

cwcli export Commands

The following table lists the command part of the **cwcli export** syntax.

Command	Description
cwcli export changeaudit	Generates Change Audit log data in XML format.
cwcli export config	Generates configlets in XML format
cwcli export inventory	Generates Inventory data in XML format.

You must invoke the **cwcli export** command with one of the core commands specified in the above table. If no core command is specified, **cwcli export** can execute the **-v** or **-h**. arguments only. Argument **-v** specifies the version of the **cwcli export** utility and argument **-h** (or null argument) displays the usage information of this tool.

cwcli export Global Arguments

The following describes the mandatory and optional global arguments for **`cwcli export`**:

Global Arguments	Description
`-u` *userid*	Mandatory Specifies the LMS username.
`-p` *password*	Mandatory Specifies the password for the LMS username. If you want to avoid the **`-p`** argument which will reveal the password in clear text in cli, you will have to store your username and password in a file and set a variable `cwcli` *`CWCLIFILE`* which points to the file. You will have to maintain this file and control access permissions to prevent unauthorized access. **`cwcli export`** looks for current working directory if `cwcli` *`CWCLIFILE`*is set only to file name instead of full path. If you use the **-p** argument, even after setting the `cwcli` *`CWCLIFILE`* variable the password is taken from the command line instead of `cwcli` *`CWCLIFILE`*. This is not secure and usage of this argument is not recommended. The password must be provided in the file in the following format: username password where username is the LMS user name given in the command line.The delimiter between the username and password is single blank space. You must enter the delimiter if the password is blank. Otherwise, **`cwcli export`** will fail to validate the password.The password file can contain multiple entries with different user names. The password that matches first is considered in case of duplicate entries. **Note** If **`-p`** *password* is used, the password is read from the command line instead of `cwcli` *`CWCLIFILE`*. This is highly insecure and therefore not recommended. See Setting CWCLIFILE Environment Variable for more details.
{ **`-device`** *devicename* | **`-view`** *viewname*| **`-input`** *inputfilename* | **`-ipaddress`** *mgmt-ip-address* }	Mandatory **`-device`** *devicename* Specifies the device name of the device that you have added in the Device and Credentials database (Inventory > Device Administration > Add / Import / Manage Devices). You can enter multiple device name separated by a comma. You can use either wildcard or specific device(s) but not at the same time. The argument syntax used for **`-device`** argument may be a single device or a device list. Devices in a list are separated by a '**,**'. The wild card symbol '**%**' may be used to specify a group of devices having a pattern. For example if a pattern ***x%*** is specified as a device in the list, then all the LMS devices that have names that start with x will be selected for this operation.

Global Arguments	Description
{ **-device** *devicename* | **-view** *viewname* | **-input** *inputfilename* | **-ipaddress** *mgmt-ip-address*}	Mandatory **-view** *viewname* If the data needs to be generated for all the devices in a specific group, you can use the **-view** argument. You can use this argument to generate data for devices in all device views including system-defined groups and user-defined groups. You can enter multiple group name separated using a comma. For view name, you have to enter the fully qualified path as in the Group Administration window. To separate the path you must use forward slash only. For example, **-view** "/RME@*ciscoworks_servername*/All Devices"
{ **-device** *devicename* | **-view** *viewname* **-input** *inputfilename* | **-ipaddress** *mgmt-ip-address*}	Mandatory **-input** *inputfilename* You can create an input list file that contains a list of devices to perform the operation on. The contents of the input list file are a sequence of lines. Each line specifies a device name as entered in the Device and Credential Repository. The arguments must be specific to the function. You cannot include group names in the input list file. You can include comments in the input list file by starting each commented line with #. The input file should be of this format: **-device** 1.1.1.1,2.2.2.2,3.3.3.3 or **-device** 1.1.1.1 **-device** 2.2.2.2 **-device** 3.3.3.3
{ **-device** *devicename* | **-view** *viewname* **-input** *inputfilename* | **-ipaddress** *mgmt-ip-address*}	Mandatory **-ipaddress** *mgmt-ip-address* Specify the device IP4 address as entered in the Device and Credential Repository. You can enter multiple IP address with comma separated. You cannot use this option with **-device**, **-view**, or **-input**. Also, you cannot specify wildcard characters.
-d *debuglevel*	Optional *debug_level* is a number between 1 (the least information is sent to the debug output) and 5 (the most information is sent to the debug output). If you do not specify this argument, 4(INFO) is the default debug level.
-l *logfile*	Optional Logs the results of the **cwcli export** command to the specified log file name. By default the command output will be displayed on the standard out.

Global Arguments	Description
`-m` mailid	Optional Mails the results of the `cwcli export` command to the specified email address. This argument notifies you whether the task is completed. Only one mail id can be given at a time.
`-f` *filename*	This is applicable only for changeaudit and inventory applications. Optional Specifies the name of the file to which the changeaudit and inventory information is to be exported on LMS server. If you are using `cwcli` remotely (get or post request), by default the output file is available at this location on LMS server: On Windows: *NMSROOT*\MDC\tomcat Where, *NMSROOT* is the LMS installed directory. On Solaris and Soft Appliance: *NMSROOT*/objects/dmgt

Archiving cwcli export Data in XML File

By default, the data generated through `cwcli export` command is archived at the location:

cwcli export *Command*	Location on LMS Server
changeaudit	On Solaris and Soft Appliance: /var/adm/CSCOpx/files/rme/archive/*YYYY-MM-DD-HH-MM-SS*-changeaudit.xml
	On Windows: *NMSROOT*\files\rme\archive*YYYY-MM-DD-HH-MM-SS*-changeaudit.xml
config	On Solaris and Soft Appliance: /var/adm/CSCOpx/files/rme/cwconfig/*YYYY-MM-DD-HH-MM-SS-MSMSMS-Device_Display_Name*.xml
	On Windows: *NMSROOT*\files\rme\cwconfig*YYYY-MM-DD-HH-MM-SS-MSMSMS-Device_Display_Name*.xml
inventory	On Solaris and Soft Appliance: /var/adm/CSCOpx/files/rme/archive/*YYYY-MM-DD-HH-MM-SS*-inventory.xml
	On Windows: *NMSROOT*\files\rme\archive*YYYY-MM-DD-HH-MM-SS*-inventory.xml

Where *NMSROOT* is the LMS installed directory.

You can also specify a directory to store the output. This application does not have a feature to delete the files created in the archive. You should delete the files when necessary.

While generating data through the servlet, the output will be displayed in the client terminal.

Running cwcli export changeaudit

Using this command you can export the Change Audit log data in the XML format.

The command syntax for **cwcli export changeaudit** is:

cwcli export changeaudit {**-u** *username* **-p** *password* **-device** *devicenames*}
[**- ipaddress** *mgmt-ip-address*]

[**-f** *filename*]

[**-from** *mm/dd/yyyy*] ---> eg: 05/01/2004

[**-to** *mm/dd/yyyy*] ---> eg: 05/06/2004

[**-app** *comma separated list of applications*]

[**-cat** *comma separated list of categories*]

Arguments in square brackets ([]) are optional; arguments in curly braces ({}) are required. You must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).

If you enter an argument which has space then use double quotes for that argument. For example for Software Management, you enter this as “Software Management”.

The following table describes the arguments that are specific to **cwcli export changeaudit** command. The other common arguments used by **cwcli export** are explained in Using the cwcli export Command.

Arguments	Description
[**-from** *mm/dd/yyyy*]	Optional. Enter the from date. If you enter only **-from** date then the report will be generated from the date that you have specified, to the current date.
[**-to** *mm/dd/yyyy*]	Optional. Enter the from date. If you enter only **-to** date then the report will be generated from the date LMS has logged Change Audit record to the **-to** date that you have specified.

Arguments	Description
[**-app** *comma separated list of applications*]	Specify the application name. The supported applications are: • Archive Mgmt • ConfigEditor • CwConfig • ICServer • NetConfig • Software Management If you do not specify the **-app** argument, then changes made by all applications is reported.
[**-cat** *comma separated list of categories*]	Specify the category name. The supported categories are: • CONFIG_CHANGE—Configuration changes on the device. • INVENTORY_CHANGE—Hardware changes on the device. • SOFTWARE_CHANGE—Software changes on the device. If you do not specify the **-cat** argument, then changes made by all categories is reported.

Note If you do not enter **-from** and **-to** arguments, all the Change Audit records logged till the current date will be displayed.

The following sections describes:

- XML Schema for cwcli export changeaudit
- XML Schema for Configuration Management Application
- XML Schema for Software Management
- Usage Examples for cwcli export changeaudit Command

XML Schema for cwcli export changeaudit

The following is the schema used for exporting the change audit data in XML format.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2000/10/XMLSchema-->
<xsd:schema xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
 <xsd:element name = "changeRecord">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref = "groupId"/>
    <xsd:element ref = "category"/>
    <xsd:element ref = "host"/>
    <xsd:element ref = "user"/>
    <xsd:element ref = "device"/>
    <xsd:element ref = "connectionMode"/>
    <xsd:element ref = "timestamp"/>
    <xsd:element ref = "description"/>
   </xsd:sequence>
   <xsd:attribute name = "id" use = "required" type = "xsd:integer"/>
  </xsd:complexType>
```

```
 </xsd:element>
 <xsd:element name = "groupId" type = "xsd:string"/>
 <xsd:element name = "category" type = "xsd:string"/>
 <xsd:element name = "application" type = "xsd:string"/>
 <xsd:element name = "host" type = "xsd:string"/>
 <xsd:element name = "user" type = "xsd:string"/>
 <xsd:element name = "device" type = "xsd:string"/>
 <xsd:element name = "connectionMode" type = "xsd:string"/>
 <xsd:element name = "timestamp" type = "xsd:string"/>
 <xsd:element name = "description">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref = "summary"/>
    <xsd:element ref = "details"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name = "summary" type = "xsd:string"/>
 <xsd:element name = "details">
  <xsd:complexType/>
 </xsd:element>
 <xsd:element name = "changeRecordSet">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref = "changeRecord" maxOccurs = "unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Detailed Description of changeaudit Schema

The table below describes elements in the changeaudit schema:

Field	Description
Category	Type of the change. For example, configuration, inventory, or software.
Application	Name of the LMS application involved in the network change (Device Configuration, Inventory, or Software Management).
Host	Host device from which the user accessed the device or the host name of the LMS server.
User	Name of the person who performed the change. This is the name entered when the person logged in. It can be the name under which the LMS is running, or the name under which the Telnet connection is established.
Device	Network device on which the change occurred. The device name as entered in the Device and Credential Repository.
ConnectionMode	Connection mode through which the change was made, for example, Telnet, snmp, console, or LMS.

Field	Description
Timestamp	Date and time at which the application communicated the network change or when Change Audit saw the change record.
Description	Contains the detailed information of the changes that had occurred on the device. The description changes based on the application you have selected: • Archive Mgmt (See XML Schema for Configuration Management Application for more information.) • ConfigEditor (See XML Schema for Configuration Management Application for more information.) • CwConfig (See XML Schema for Configuration Management Application for more information.) • ICServer—Inventory Collection Service (See XML Schema for Inventory Collection Service for more information.) • NetConfig (See XML Schema for Configuration Management Application for more information.) • Software Management (See XML Schema for Software Management for more information.)

The following section describes the schema used by these application when you run the command **`cwcli export changeaudit`** with **`-app`** argument:

- Archive Mgmt, ConfigEditor, CwConfig, NetConfig
- Inventory Collection Service
- Software Management

XML Schema for Configuration Management Application

The following XML schema is used by all Configuration Management application when you run the **`cwcli export changeaudit`** command with **`-app`** argument and the **`-app`** argument values as either *Archive Mgmt*, *ConfigEditor*, *CwConfig*, or *NetConfig*.

The schema file is:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--  edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by Cisco (t) -->
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    - <xs:element name="ConfigDiff">
        - <xs:annotation>
            <xs:documentation>Comment describing your root element</xs:documentation>
          </xs:annotation>
        - <xs:complexType>
            - <xs:sequence>
                - <xs:element name="OldConfig">
                    - <xs:complexType>
                        - <xs:sequence>
                            <xs:element name="Command" maxOccurs="unbounded" />
                        </xs:sequence>
                        <xs:attribute name="Version" type="xs:integer" />
                    </xs:complexType>
                </xs:element>
    - <xs:element name="NewConfig">
        - <xs:complexType>
```

```
            - <xs:sequence>
                <xs:element name="Command" maxOccurs="unbounded" />
              </xs:sequence>
                          <xs:attribute name="Vesrion" type="xs:integer" />
                      </xs:complexType>
                  </xs:element>
              </xs:sequence>
              <xs:attribute name="CASLogID" type="xs:integer" />
              <xs:attribute name="DeviceName" type="xs:string" />
          </xs:complexType>
      </xs:element>
</xs:schema>
```

Detailed Description of Configuration Management Schema

The table below describes elements in the Configuration Management schema.

Field	Description
Category	Type of the change. For example, configuration, inventory, or software.
Host	Host device from which the user accessed the device or the host name of the LMS server.
Application	Name of the application. For example, Archive Mgmt, NetConfig, etc.
User	Name of the person who performed the change. This is the name entered when the person logged in. It can be the name under which the LMS is running, or the name under which the Telnet connection is established.
Device	Network device on which the change occurred. The device name as entered in the Device and Credential Repository.
ConnectionMode	Connection mode through which the change was made, for example, Telnet, snmp, console, or LMS.
Timestamp	Date and time at which the application communicated the network change or when Change Audit saw the change record.
Summary	Description describing what caused the change. For example: • Configuration Download due to job • Syslog triggered Config Collection
ConfigDiff	• FirstConfig, SecondConfig • DeviceName—Network device on which the change occurred. The device name as entered in the Device and Credential Repository. • Version—Configuration file version number. • CommandDiff Polarity—Changes in the configuration file. – POSNEG—No change – POSITIVE —Added new commands – IGNORED—Ignored the commands – NEGATIVE—Deleted the commands

XML Schema for Inventory Collection Service

The following XML schema is used by Inventory Collection Service application when you run the **cwcli export changeaudit** command with **-app** argument and the **-app** argument values as ICServer.

The schema file is:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
<xsd:element name = "InventoryChangeDetailRecord">
    <xsd:complexType>
        <xsd:sequence maxOccurs = "unbounded">
            <xsd:element ref = "Section"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "Section">
    <xsd:complexType>
        <xsd:sequence maxOccurs = "unbounded">
            <xsd:element ref = "Attributes"/>
        </xsd:sequence>
        <xsd:attribute name = "Name" type = "xsd:string"/>
        <xsd:attribute name = "Identity" type = "xsd:string"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "Attributes">
    <xsd:complexType>
        <xsd:all maxOccurs = "unbounded">
            <xsd:element ref = "Previous_value"/>
            <xsd:element ref = "Current_value"/>
            <xsd:element ref = "Action"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
    <xsd:element name = "Previous_value" type = "xsd:string"/>
    <xsd:element name = "Current_value" type = "xsd:string"/>
    <xsd:element name = "Action" type = "xsd:string"/>
</xsd:schema>
```

Detailed Description of Inventory Collection Service Schema

The table below describes elements in the Inventory Collection Service schema.

Field	Description
Name	Name of the physical and logical entities. The main physical entities are Chassis, Backplane, Card, CommunicationConnector, FlashDevice, FlashPartition, FlashFile, SoftwareIdentity, PhysicalMemory The main logical entities are ManagedNetworkElement, LogicalModule, Port, MemoryPool, OSElement, IPProtocolEndpoint, IfEntry, AdditionalInformation See Detailed Description of the Inventory Schema for further information.
Identity	Identifies the entity that has changed on the device. For example: If the Flash File name has changed then Identity will be Flash Device 2, Flash Partition 3.

Field	Description
AttributeName	Name of the different physical and logical entities For example: In MemoryPool, the different entities are User, Free, PoolType. See Detailed Description of the Inventory Schema for further information.
Previous_value	Value of the entity before the change occurred.
Current_value	Value of the entity after the change occurred.
Action	Type of change that has occurred on the device: • Inserted— Added a new entity. • Changed—Changed in the entity. • Deleted—Deleted an entity.

XML Schema for Software Management

The following XML schema is used by Software Management application when you run the **`cwcli export changeaudit`** command with **`-app`** argument and the **`-app`** argument values as Software Management.

The schema file is:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--  edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by Cisco -->
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
- <xs:element name="SwimHistoryRecord">
    - <xs:annotation>
        <xs:documentation>Models a set of image changes on the device.</xs:documentation>
    </xs:annotation>
- <xs:complexType>
    - <xs:sequence>
        - <xs:element name="JobID" type="xs:positiveInteger" minOccurs="0">
    - <xs:annotation>
        <xs:documentation>ID of the Job in which the change happened</xs:documentation>
    </xs:annotation>
    </xs:element>
    - <xs:element name="ImageChange" maxOccurs="unbounded">
- <xs:complexType>
    - <xs:sequence>
          <xs:element name="OldImage" type="Image" />
          <xs:element name="NewImage" type="Image" />
      </xs:sequence>
      <xs:attribute name="ID" type="xs:positiveInteger" use="required" />
      </xs:complexType>
      </xs:element>
      </xs:sequence>
  </xs:complexType>
</xs:element>
- <xs:complexType name="Image">
- <xs:annotation>
     <xs:documentation>Models an Image.</xs:documentation>
  </xs:annotation>
- <xs:sequence>
    - <xs:element name="Type">
        - <xs:annotation>
          <xs:documentation>Label of corresponding image type from enumeration
        com.cisco.nm.xms.xdi.ags.imageparser.ImageType</xs:documentation>
          </xs:annotation>
```

```
        - <xs:simpleType>
            - <xs:restriction base="xs:string">
            <xs:whiteSpace value="preserve" />
          </xs:restriction>
        </xs:simpleType>
        </xs:element>
      <xs:element name="Name" type="xs:string" />
      <xs:element name="Version" type="xs:string" />
- <xs:element name="Attribute" minOccurs="0" maxOccurs="unbounded">
    - <xs:complexType>
    - <xs:sequence>
        - <xs:element name="AttributeName">
        - <xs:simpleType>
            - <xs:restriction base="xs:string">
            <xs:whiteSpace value="preserve" />
      </xs:restriction>
      </xs:simpleType>
  </xs:element>
- <xs:element name="AttributeValue">
    - <xs:simpleType>
        - <xs:restriction base="xs:string">
         <xs:whiteSpace value="preserve" />
      </xs:restriction>
      </xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

Detailed Description of Software Management Schema

The table below describes elements in the Software Management schema.

Field	Description
Category	Type of the change. For example, configuration, inventory, or software.
Host	Host device from which the user accessed the device or the host name of the LMS server.
Application	Name of the application. For example, Archive Mgmt, NetConfig, etc.
User	Name of the person who performed the change. This is the name entered when the person logged in. It can be the name under which the LMS is running, or the name under which the Telnet connection is established.
Device	Network device on which the change occurred. The device name as entered in the Device and Credential Repository.
ConnectionMode	Connection mode through which the change was made, for example, Telnet, snmp, console, or LMS.
Timestamp	Date and time at which the application communicated the network change or when Change Audit saw the change record.
Summary	Description describing what caused the change. For example, Software upgrade.
Job ID	Job ID for the Software Upgrade.

Field	Description
OldImage	Displays the details on device type, name of the software image, and version of the image.
NewImage	Displays the details on device type, name of the software image, and version of the image.

Usage Examples for cwcli export changeaudit Command

This section provides some examples of usage for the **cwcli export changeaudit** command.

Example 1: To generate the Change Audit report for all applications and categories for a particular device group.

cwcli export changeaudit -u *admin* **-p** *admin* **-view** *"/RME@ciscoworksservername/Normal Devices*"

```
SUMMARY
========
        Successful: export:
D:/PROGRA~1/CSCOpx/files/rme/archive/2004-10-15-04-09-42-changeaudit.xml
```

Example 2: To generate the Change Audit report for a specific application and category for a device in a given time frame

cwcli export changeaudit -u *admin* **-p** *admin* **-device** *10.6.8.6* **-from** *09/30/2004* **-to** *10/15/2004* **-app** "*Archive Mgmt*" **-cat** *CONFIG_CHANGE*

```
SUMMARY
========
        Successful: export:
D:/PROGRA~1/CSCOpx/files/rme/archive/2004-10-15-04-44-50-changeaudit.xml
```

Example 3: To generate the Change Audit report in the given output file

cwcli export changeaudit -u *admin* **-p** *admin* **-device** *%* **-f** *changeaudit.xml*

```
SUMMARY
========
        Successful: export: changeaudit.xml
```

The output for this is written to the changeaudit.xml file in the *Install_dir*/CSCOpx/bin directory. Where *Install_dir* is the LMS installed directory.

Example 4: To generate the Change Audit using the cwcli get request

The password that you enter here must be in base64 encoded.

In this example,

- *YWRtaW4=* is the base64 encoded password for admin.
- %25 is the URL encode for "%"
- %2f is the URL encode for "_"

Enter this in your browser:

http://*ciscowork_servername*:1741/rme/cwcli?command=cwcli export changeaudit -u *admin* -p *YWRtaW4=* -device *10.7.3.8* -app *%22Archive Mgmt%22* -cat *%22CONFIG%2fCHANGE%22* -f *changeaudit.xml*

The output for this is written to the changeaudit.xml file.The changeaudit.xml file is located:

On Windows:

NMSROOT\MDC\tomcat

Where, *NMSROOT* is the LMS installed directory.

On Solaris and Soft Appliance:

NMSROOT/objects/dmgt

Example 5: To generate the Change Audit report using cwcli post request method

The password that you enter here must be in base64 encoded. In this example, *YWRtaW4=* is the base64 encoded password for admin.

The payload file, *changeaudit.xml* contains:

```
<payload>
   <command>
cwcli export changeaudit -u admin -p YWRtaW4= -device 1.6.8.6 -from 09/30/2004 -app
"Archive Mgmt" -cat CONFIG_CHANGE -view "/RME@CiscoWorks_servername/Pre-deployed" -f
changeauditreport.xml
   </command>
</payload>
```

At the command prompt enter:

perl *samplepost.pl* **https://LMS_Servername:443/rme/cwcli** *changeaudit.xml*

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

```
SUMMARY
========
        Successful: export: changeauditreport.xml
<!-- Processing complete -->
```

The output for this is written to the *changeauditreport.xml* file.The *changeauditreport.xml* file is located:

On Windows:

NMSROOT\MDC\tomcat

Where, *NMSROOT* is the LMS installed directory.

On Solaris and Soft Appliance:

NMSROOT/objects/dmgt

Running cwcli export config

Using this command you can retrieve the configuration data in the XML format specified by the schema. The Configlet Generator provides a wrapper over the existing Config Archive to retrieve configlets data for the selected device. The exported data contains the entire running configuration data.

The command syntax for **cwcli export config** is:

cwcli export config{**-u** *username* **-p** *password*} [**-d** *debuglevel*] [**-m** *mailid*] [**-l** *logfile*] {**-device** *devicename* | **-input** *inputfilename* | **-view** *viewname* | **- ipaddress** *mgmt-ip-address*}

Arguments in square brackets ([]) are optional; arguments in curly braces ({}) are required. You must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).

If you enter an argument which has space then use double quotes for that argument.

The following table describes the argument that is specific to **cwcli export config** command. The other common arguments used by **cwcli export** are explained in Using the cwcli export Command.

Arguments	Description
-s 1	Optional. Displays the exported configuration file on the console. If you use this command, you can specify only one device. You cannot export the configuration files of multiple devices. To export the configuration files of multiple devices, either make multiple requests to the servlet, or get these files from the LMS server. Usage of this option: **cwcli export config -u** *admin* **-p** *admin* **-device** 10.22.33.44 **-s 1**

The output files depends on the number of devices specified. There are as many configuration XML output files as the number of devices. The output files are created under this location on LMS server:

On Solaris and Soft Appliance:
/var/adm/CSCOpx/files/rme/cwconfig/*YYYY-MM-DD-HH-MM-SS-XXX-Device_Display_Name.xml*

On Windows:
NMSROOT\files\rme\cwconfig*YYYY-MM-DD-HH-MM-SS-XXX-Device_Display_Name.xml*

Where *NMSROOT* is the LMS installed directory.

XML Schema for cwcli export config

The following is the schema used for exporting the configuration data in XML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Cisco -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="DeviceConfiguration">
<xs:annotation>
<xs:documentation>This has list of Configlets</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="Confilglet" maxOccurs="unbounded"/>
<xs:element name="DeviceName" type="xs:string">
<xs:annotation>
<xs:documentation>Device Name as managed by RME</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="DeviceFamily" type="xs:string">
<xs:annotation>
<xs:documentation>MDF devcie family</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="CreationTime" type="xs:date">
<xs:annotation>
<xs:documentation>Date and Time this was created</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Version" type="xs:string">
<xs:annotation>
<xs:documentation>Config File Version</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Data" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Confilglet">
<xs:annotation>
<xs:documentation>Configlet can have subconfiglets</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="Confilglet" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="command" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="LineNo"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="SubModeCommand" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Command to change the mode</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
<xs:attribute name="Name" type="xs:string" use="required">
<xs:annotation>
<xs:documentation>Configlet Name</xs:documentation>
```

```
</xs:annotation>
</xs:attribute>
<xs:attribute name="Checkedout" type="xs:boolean" use="optional" default="false">
<xs:annotation>
<xs:documentation>Future Use </xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="Rebuild" type="xs:boolean" use="optional" default="false">
<xs:annotation>
<xs:documentation>Specifies if the entire list of commands in particular configlet needs a rebuild if any of
the coammnds is modified</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="Submode" type="xs:boolean" use="optional" default="false">
<xs:annotation>
<xs:documentation>Specfies if the commands under the configlet falls under a submode</xs:documentation>
</xs:annotation>
</xs:attribute>
<xs:attribute name="OrderSensitive" type="xs:boolean" use="optional" default="false">
<xs:annotation>
<xs:documentation>Specifies if the commands in the configlet are oreder sensitive or not</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:complexType>
</xs:element>
</xs:schema>
```

Detailed Description of config Schema

The table below describes elements in the config schema:

Field	Description
Device	Device device name as entered in the Device and Credential Repository.
Date	Date and time at which the configuration changes have occurred.
Version	Configuration file version.
Configlet name	Name of the configlet. The available configlets vary from device to device; the following are examples: • SNMP displays SNMP configuration commands, for example, snmp-server community public RO. • IP Routing displays IP routing configuration commands, for example, router abcd 100. • Interface folder displays the different interface configuration commands, for example, Interface Ethernet0 and Interface TokenRing. • Global displays global configuration commands, for example no ip address. • Line con 0 displays configuration commands for line console 0. • IP displays IP configuration commands, for example, **`ip http server`**.

Usage Examples for cwcli export config Command

This section provides some examples of usage for the **cwcli export config** command.

Example 1: To generate the config report for a particular device group

cwcli export config -u *admin* **-p** *admin* **-view** *"/RME@ciscoworksservername/Normal Devices*"

```
SUMMARY
========
        Successful: ConfigExport:D:/PROGRA~1/CSCOpx/files/rme/cwconfig
```

The output folder will contain separate config file for every devices in the Normal Devices group.

Example 2: To generate the config report for the devices that are specified in the device input file

The input configdevices.txt contains these devices:

-device 10.22.33.44,10.22.33.55,1.1.1.1

cwcli export config -u *admin* **-p** *admin* **-input** *configdevices.txt*

Example 3: To generate the config using the cwcli get request

The password that you enter here must be in base64 encoded.

In this example,

- *YWRtaW4=* is the base64 encoded password for admin.
- %25 is the URL encode for "*%*"

Enter this in your browser:

http://*ciscowork_servername***:1741/rme/cwcli?command=cwcli export config -u** *admin* **-p** *YWRtaW4=* **-device** *%25*

```
<!-- Processing Starts  -->
SUMMARY
========
Successful: ConfigExport: D:/PROGRA~1/CSCOpx/files/rme/cwconfig
<!-- Processing complete -->
```

Example 4: To generate the Change Audit report using cwcli post request method

The password that you enter here must be in base64 encoded. In this example, *YWRtaW4=* is the base64 encoded password for admin.

The payload file, *config.xml* contains:

```
<payload>
   <command>
cwcli export config -u admin -p YWRtaW4= -device 1.6.8.6
   </command>
</payload>
```

At the command prompt enter:

perl *samplepost.pl* **https://LMS_Servername:443/rme/cwcli** *config.xml*

```
<!-- Processing Starts  -->
```

```
SUMMARY
========
        Successful: ConfigExport: D:/PROGRA~1/CSCOpx/files/rme/cwconfig
<!-- Processing complete -->
```

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

Running cwcli export inventory Command

Using this command you can export inventory data in the XML format.

The command syntax for **cwcli export inventory** is:

cwcli export inventory {**-u** *username* **-p** *password*}[**-d** *debuglevel*] [**-m** *mailid*] [-l *logfile*] [**-f** *filename*] {**-device** *devicename* | **-input** *inputfilename* | **-view** *viewname* | **- ipaddress** *mgmt-ip-address*} [**-hop** *hopdevice*]

The above command retrieves the inventory data in XML format specified by the schema. The **-f** parameter stores the output in the file specified by *filename*. If you have not specified the filename, the output is stored at the following location:

PX_DATADIR/rme/archive/timestampinventory.xml (On Solaris and Soft Appliance)

PX_DATADIR\rme\archive\timestampinventory.xml (On Windows)

Where *PX_DATADIR* is the *NMSROOT*/files directory and *NMSROOT* is the LMS installed directory.

The device name can also have a wild card symbol "**%**" to choose all devices with that particular name.

If the number of devices is large, the list of devices can be stored in an input file and the name of the input file can be given in the command line.The input argument cannot occur with the device or view arguments.

If the data needs to be generated for all the devices in a specific group, you can use the **-view** argument. You can use this argument to generate data for devices in all device groups including system-defined groups and user-defined groups.

The following table describes the arguments that are specific to **cwcli export inventory** command. The other common arguments used by **cwcli export** are explained in Using the cwcli export Command.

Global Arguments	Description
-hop *hopdevice*	Optional Used to increase performance by using more memory. This indicates the number of devices to be worked upon at a time. By default, this value is 1.

Given below is the list of combinations, which could occur for the inventory command.

cwcli export inventory -u *admin* **-p** *admin* **-f** *myinv.xml*

cwcli export inventory -u *admin* **-p** *admin* **-f** *myinv.xml* **-device** *device1*

cwcli export inventory -u *admin* **-p** *admin* **-device** *device%*

cwcli export inventory -u *admin* **-p** *admin* **-input** *inv.txt*

cwcli export inventory -u *admin* **-p** *admin* **-view** *"/RME@ciscoworksservername/Normal Devices"*

cwcli export inventory -u *admin* **-p** *admin* **-f** *myinv.xml* **-input** *inv.txt*

To apply the **cwcli export** command on more than one LMS device you must use the format in the example given below. The parameter, inputlist, is a text file which will have the list of device names separated by a new line. A line starting with # will be treated as a comment.

Example:

```
#comment
-device device1,device2,device3
#comment
```

where device1, device2, and device3 are device displaynames.

XML Schema for cwcli export inventory Data

The following is the schema used for exporting the inventory data in XML format.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema" elementFormDefault = "qualified"
attributeFormDefault = "unqualified">
        <!--This schema is based on the classes defined in Cisco Information Model V2.0 (CIMCXV2.0)
Each Device has Chassis and NetworkElement.
Chassis:
        Chassis contains a blackplane and multiple Cards. Each Card contains CommunicationConnectors and
multiple daughter cards. Flash Devices reside on the Cards.
NetworkElement:
System Information, Interface Information and LogicalModules. LogicalModules contain OSElements and Logical
Ports.
The element AdditionalInformation is meant to capture device specific details that are not part of the common
schema.
-->
<xs:element name = "InvDetails">
<xs:complexType>
<xs:sequence>
<xs:element ref = "SchemaInfo" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "RMEPlatform" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "SchemaInfo">
<xs:complexType>
<xs:sequence>
<xs:element name = "RMEServer" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "CreatedAt" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "SchemaVersion" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "RMEPlatform">
<xs:complexType>
<xs:sequence>
<xs:element ref = "Cisco_Chassis" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_NetworkElement" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_ComputerSystemPackage" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref= "Cisco_EnergyWise" minOccurs="0" maxOccurs= "1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_Chassis">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
```

```
<xs:element name = "Model" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "HardwareVersion" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "SerialNumber" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "ChassisSystemType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NumberOfSlots" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NoOfCommunicationConnectors" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "Cisco_Backplane" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_Card" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_Backplane">
<xs:complexType>
<xs:sequence>
<xs:element name = "BackplaneType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Model" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "SerialNumber" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_Card">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "RequiresDaughterBoard" type = "xs:boolean" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Model" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "SerialNumber" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "LocationWithinContainer" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PartNumber" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "CardType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "HardwareVersion" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Description" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "OperationalStatus" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "FWManufacturer" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Manufacturer" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NumberOfSlots" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NoOfCommunicationConnectors" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "SoftwareIdentity" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_CommunicationConnector" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_FlashDevice" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_PhysicalMemory" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_Card" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_CommunicationConnector">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "ConnectorType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Description" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "POEAdminStatus" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "MaximumPower" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PowerAllocated" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_FlashDevice">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name = "InstanceID" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "FlashDeviceType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Size" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NumberOfPartitions" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "ChipCount" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Description" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Removable" type = "xs:boolean" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "Cisco_FlashPartition" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_FlashPartition">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Upgrade" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NeedsErasure" type = "xs:boolean" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PartitionStatus" minOccurs = "0" maxOccurs = "1">
<xs:simpleType>
<xs:restriction base = "xs:string">
<xs:enumeration value = "unknown"/>
<xs:enumeration value = "readOnly"/>
<xs:enumeration value = "runFromFlash"/>
<xs:enumeration value = "readWrite"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name = "FileSystemSize" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "AvailableSpace" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "FileCount" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "Cisco_FlashFile" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_FlashFile">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "FileSize" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "FileStatus" minOccurs = "0" maxOccurs = "1">
<xs:simpleType>
<xs:restriction base = "xs:string">
<xs:enumeration value = "unknown"/>
<xs:enumeration value = "deleted"/>
<xs:enumeration value = "invalidChecksum"/>
<xs:enumeration value = "valid"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name = "Checksum" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_PhysicalMemory">
<xs:complexType>
<xs:sequence>
<xs:element name = "MemoryType" minOccurs = "0" maxOccurs = "1">
<xs:simpleType>
```

```
<xs:restriction base = "xs:string">
<xs:enumeration value = "nvRam"/>
<xs:enumeration value = "NVRAM"/>
<xs:enumeration value = "processorRam"/>
<xs:enumeration value = "RAM"/>
<xs:enumeration value = "ROM"/>
<xs:enumeration value = "FEPROM"/>
<xs:enumeration value = "BRAM"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name = "Capacity" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_NetworkElement">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:integer" maxOccurs = "1"/>
<xs:element name = "Description" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PrimaryOwnerName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PhysicalPosition" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "SysObjectId" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "SysUpTime" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "OfficialHostName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NumberOfPorts" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "Cisco_LogicalModule" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_Port" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_MemoryPool" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_IfEntry" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_IPProtocolEndpoint" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_PEHasIfEntry" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_LogicalModule">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "ModuleNumber" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "ModuleType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "EnabledStatus" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NumberOfPorts" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "Cisco_Port" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_LogicalModule" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "Cisco_OSElement" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_Port">
<xs:complexType>
<xs:sequence>
<xs:element name = "PortNumber" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PortType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "IfInstanceID" type = "xs:integer" minOccurs = "0" maxOccurs = "unbounded"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
```

```
</xs:element>
<xs:element name = "Cisco_MemoryPool">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PoolType" type = "xs:integer" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "DynamicPoolType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "AlternatePoolType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "IsValid" type = "xs:boolean" minOcurs = "0" maxOccurs = "1"/>
<xs:element name = "Allocated" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Free" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "LargestFree" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
<!--PoolType ValueMap  {"0", "1", "2", "3", "4", "5", "65536"},
Values {"Unknown", "Processor", "I/O", "PCI", "Fast", "Multibus", "Dynamic"},
-->
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_OSElement">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "OSFamily" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Version" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Description" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_IfEntry">
<xs:complexType>
<xs:sequence>
<xs:element name = "InstanceID" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "InstanceName" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "ProtocolType" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Speed" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "RequestedStatus" minOccurs = "0" maxOccurs = "1">
<xs:simpleType>
<xs:restriction base = "xs:string">
<xs:enumeration value = "up"/>
<xs:enumeration value = "down"/>
<xs:enumeration value = "testing"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name = "OperationalStatus" minOccurs = "0" maxOccurs = "1">
<xs:simpleType>
<xs:restriction base = "xs:string">
<xs:enumeration value = "Up"/>
<xs:enumeration value = "Down"/>
<xs:enumeration value = "Testing"/>
<xs:enumeration value = "Unknown"/>
<xs:enumeration value = "Dormant"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name = "Description" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "PhysicalAddress" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "NetworkAddress" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name = "Cisco_IPProtocolEndpoint">
<xs:complexType>
<xs:sequence>
<xs:element name = "Address" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "SubnetMask" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "DefaultGateway" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element ref = "AdditionalInformation" minOccurs = "0" maxOccurs = "unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_PEHasIfEntry">
<xs:complexType>
<xs:sequence>
<xs:element name = "Cisco_IPProtocolEndpoint" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Cisco_IfEntry" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "Cisco_ComputerSystemPackage">
<xs:complexType>
<xs:sequence>
<xs:element name = "Antecedent" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<xs:element name = "Dependent" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
<!--
Antecedent is the InstanceID from Cisco_Chassis Element
Dependent is the InstanceID from Cisco_NetworkElement
-->
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Cisco_EnergyWise">
<xs:complexType>
<xs:sequence>
<xs:element name= "InstanceID" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element name= "DomainName" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element name= "Role" type= "xs:string" minOccurs= "0" maxOccurs="1"/>
<xs:element name= "Keyword" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element name= "Importance" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element ref= "Cisco_EnergyWiseInterface" minOccurs= "0" maxOccurs= "1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name= "Cisco_EnergyWiseInterface">
<xs:complexType>
<xs:sequence>
<xs:element name= "InstanceID" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element name= "Description" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element name= "Role" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element name= "Keyword" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
<xs:element name= "Importance" type= "xs:string" minOccurs= "0" maxOccurs= "1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "SoftwareIdentity">
<xs:complexType>
<xs:sequence>
<xs:element name = "Classification" minOccurs = "0" maxOccurs = "1">
<xs:simpleType>
<xs:restriction base = "xs:string">
<xs:enumeration value = "Firmware"/>
<xs:enumeration value = "Software"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

```
<xs:element name = "VersionString" type = "xs:string" minOccurs = "0" maxOccurs = "1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name = "AdditionalInformation">
<xs:complexType>
<xs:sequence>
<xs:element name = "AD" minOccurs = "0" maxOccurs = "unbounded">
<xs:complexType>
<xs:attribute name = "name" type = "xs:string"/>
<xs:attribute name = "value" type = "xs:string"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Detailed Description of the Inventory Schema

The inventory schema provides the structure for the inventory information exported from LMS. The schema divides inventory information into two groups:

- Physical Inventory
- Logical Inventory

The Physical Inventory contains the chassis information and related details for the device. The main elements in the schema for the physical inventory part are:

- Chassis (Cisco_Chassis
- Backplane (Cisco_ Backplane
- Card (Cisco_Card)
- CommunicationConnector (Cisco_CommunicationConnector)
- FlashDevice (Cisco_FlashDevice)
- FlashPartition (Cisco_FlashPartition)
- FlashFile (Cisco_FlashFile)
- .SoftwareIdentity (Cisco_SoftwareIdentity)
- PhysicalMemory (Cisco_PhysicalMemory)

The Logical Inventory part of the schema contains the details of the managed network element. The main elements in the schema for the logical inventory part are:

- .ManagedNetworkElement (Cisco_NetworkElement)
- LogicalModule (Cisco_LogicalModule)
- Port (Cisco_Port)
- MemoryPool (Cisco_MemoryPool)
- OSElement (Cisco_OSElement)
- IPProtocolEndpoint (Cisco_IPProtocolEndpoint)
- IfEntry (Cisco_IfEntry)
- Additional Information

Chassis (Cisco_Chassis

The Chassis class represents the physical elements that enclose other elements in the device and provide specific functions, such as a desktop, networking node, UPS, disk or tape storage, or a combination of these functions.

The following table describes the elements in Chassis:

Element	Description
InstanceID	Unique identifier.
Model	Chassis model / Chassis ID.
Version	Hardware version of the chassis
SerialNumber	Serial number associated with the chassis.
Type	Chassis type.

Element	Description
NumberOfSlots	Number of slots in a chassis.
NoOfCommunicationConnectors	Number of physical connectors in a chassis.

Chassis also contains the elements Card and Backplane.

Backplane (Cisco_ Backplane

Backplane is an instance of a backplane in a chassis. The following table describes the elements in Backplane:

Element	Description
BackplaneType	Type of backplane
Model	Model of the backplane
SerialNumber	Serial number of the backplane.

Card (Cisco_Card)

Card represents:

- A type of physical container that can be plugged into another card, motherboard, or hosting board
- A motherboard or hosting board in a chassis

This element includes any package capable of carrying signals and providing a mounting point for physical components such as chips, or other physical packages such as other cards. The following table describes the elements in Card:

Element	Description
InstanceID	Card number. This is used to correlate with the logical part of the card.
Model	Model of the card.
SerialNumber	Serial number of the card.
LocationWithinContainer	Number that indicates the physical slot number. This can be used as an index into a system slot table, irrespective of whether that slot is physically occupied or not.
PartNumber	Part number of the Hardware Component.
CardType	Type of the card (Card Type)
Description	Descriptive string used to describe the card.
OperationalStatus	Status of the card describing whether it is up or down
FWManufacturer	Manufacturer of the firmware
Manufacturer	Manufacturer of the hardware
NumberOfSlots	Number of slots in the card.
NoOfCommunicationConnectors	Number of ports in the card.

Apart from the elements described in the table above, the card element also contains reference to itself, which can represents a sub card. It also contains other elements such as CommunicationConnector and FlashDevice.

CommunicationConnector (Cisco_CommunicationConnector)

CommunicationConnector represents a physical port. The table below describes the elements in CommunicationConnector:

Element	Description
InstanceID	Indicates the connector number for the chassis or system.
ConnectorType	Type of the physical port.
Description	Descriptive string used to describe the card.

FlashDevice (Cisco_FlashDevice)

FlashDevice represents physical flash memory. Flash memory may reside on a PCMCIA card, line card, or any other type of card. FlashDevice may consist of one or more actual flash memory chips.

It also consists of reference to one or more flash partitions described in FlashPartition. The table below describes the elements in FlashDevice.

Element	Description
InstanceID	FlashDevice sequence number to index the flash devices within the table of initialized FlashDevices.
InstanceName	Name of FlashDevice. This name is used to refer to the device within the system. Flash operations get directed to a device based on this name.
Size	Total size of FlashDevice. For a removable device, the size will be zero if the device has been removed.
NumberOfPartitions	Number of Flash partitions present in f FlashDevice
ChipCount	Total number of chips within FlashDevice. This element provides information to a network management station on how much chip information to expect. It also helps the management station to check the chip index against an upper limit when randomly retrieving chip information for a partition.
Description	Description of a FlashDevice. The description is meant to explain what FlashDevice is and its purpose.
Removable	Specifies whether FlashDeviceis removable. Typically, only PCMCIA Flash cards are treated as removable. Socketed Flash chips and Flash SIMM modules are not treated as removable.

FlashPartition (Cisco_FlashPartition)

FlashPartition corresponds to the Cisco-flash-mib. The elements in FlashPartiion are derived from the table of properties of ciscoFlashPartitionTable for each initialized flash partition.

When there is no explicit partitioning for a device, it is assumed that there is a single partition spanning the entire device exists. Therefore, a device always has at least one partition.

FlashPartition contains one or more FlashFileSystems as described in FlashFile. The table below describes the elements in FlashPartition.

Element	Description
InstanceID	FlashPartition sequence number used to index FlashPartitions within the table for initialized FlashPartitions.
InstanceName	FlashPartition name used to refer to a partition by the system.
PartitionStatus	Status of the partition.
Upgrade	Upgrade information for the partition. This helps to download new files into the partition, and is needed when the PartitionStatus is run from flash.
NeedsErasure	Boolean parameter indicating whether the partition requires to be erased before any write operations can occur.
Size	FlashPartition size. It should be an integral multiple of ciscoFlashDeviceMinPartitionSize. If there is a single partition, this size will be equal to ciscoFlashDeviceSize.
FreeSpace	Free space within aFlashPartition.
FileCount	Number of files stored in the file system.

FlashFile (Cisco_FlashFile)

FlashFile manages the storage and organization of files on a Flash memory device. The table below describes the elements in FlashFile

Element	Description
InstanceID	FlashFile sequence number used to index within a FlashPartition directory table.
FileSize	Size of the file in bytes.This size does not include the size of the filesystem file header.
FileStatus	Status of a file. A file could be explicitly deleted if the file system supports such a user command. Alternatively, an existing good file would be automatically deleted if another good file with the same name were copied in. Deleted files continue to occupy prime space in flash memory. A file is marked as having an invalid checksum if any checksum mismatch was detected while writing or reading the file. Incomplete files (files truncated either because of lack of free space, or because of a network download failure) are also written with a bad checksum and marked as invalid.
Checksum	File checksum stored in the file header. This checksum is computed and stored when the file is written into Flash memory, and serves to validate the data written into Flash. Where the system generates and stores the checksum internally in hexadecimal form, checksum provides the checksum in a string form. Checksum is available for all valid and invalid-checksum files.
FileName	FlashFile name as specified by the user while copying the file to Flash memory. The name should not include the colon (:) character as it is a special separator character used to separate the device name, partition name, and the file name.

.SoftwareIdentity (Cisco_SoftwareIdentity)

SoftwareIdentity provides the hardware and firmware version of the card. The table below describes elements in SoftwareIdentity.

Element	Description
Classification	Specifies whether the information is for hardware or firmware.
VersionString	Version information for software or firmware.

PhysicalMemory (Cisco_PhysicalMemory)

PhysicalMemory specifies the memory type and capacity of the device. The table below describes elements in PhysicalMemory.

Element	Description
MemoryType	Specifies the type of memory, that is whether RAM, ROM, or NVRAM.
Capacity	Capacity in bytes.

.ManagedNetworkElement (Cisco_NetworkElement)

ManagedNetworkElement is the entity that contains all logical parts of the device, which the users can configure (such as Logical Module, Port, Interfaces, Software Image details, and Memory Pool). The table below describes elements in ManagedNetworkElement.

Element	Description
InstanceID	Index number assigned to the network element.
Description	Description of the network element. This description includes the full name and version identification of the system's hardware type, operating system, and networking software. The description can have only printable ASCII characters.
PrimaryOwnerName	Identification of the contact person for this managed node, and information on how to contact this person.
InstanceName	Administratively defined name for the network element.
PhysicalPosition	Physical location of the network element.
SysObjectId	Vendor's authoritative identification of the management subsystem contained in the element.
SysUpTime	Time in hundredths of a second since the network management portion of the element was last initialized.
OfficialHostName	Name of the device.
NumberOfPorts	Number of ports in the network element.

LogicalModule (Cisco_LogicalModule)

LogicalModule is the logical device corresponding to a line card in the network device.

For example, a line card in a switch is an instance of LogicalModule, associated with the ManagedNetworkElement, in this case the switch. LogicalModule is not necessarily independently managed.

To represent a sub module, LogicalModule contains a reference to itself. It also contains Port and the OSElement. The table below describes the other elements in LogicalModule.

Element	Description
InstanceID	Index that correlates the physical card and the logical module. This helps to correlate the physical card to logical card details.
ModuleNumber	Number assigned to the module by its parent ManagedNetworkElement.
ModuleType	Type or model of the module.
InstanceName	Name of the logical module.
EnabledStatus	Status of the module, that is whether it is up or down.
NumberOfPorts	Number of ports in the logical module.

Port (Cisco_Port)

Port is the logical representation of network communications hardware - a physical connector and the setup or operation of the network chips, at the lowest layers of a network stack

For example, an Ethernet port on an Ethernet line card uses an instance of Port to represent its operational and logical properties. A port should be associated with either a LogicalModule or directly with a ManagedNetworkElement.

It also contains the element IPProtocolEndpoint. The table below describes the other elements in Port.

Element	Description
PortNumber	Number assigned to the port. Ports are often numbered relative to either a logical module or a network element.
PortType	Type of the port.
InstanceName	Name assigned to the port.
IfInstanceID	Index of the interface related to this port.

MemoryPool (Cisco_MemoryPool)

MemoryPool corresponds to entries to monitor entries. Each pool is a range of memory segregated by type or function. The table below describes the other elements in MemoryPool.

Element	Description
InstanceName	Name assigned to the MemoryPool.
PoolType	Dynamic type value assigned to a dynamic MemoryPool. This is valid only when the PoolType attribute has the value *Dynamic*. MemoryPools can be divided into two groups Predefined Pools and Dynamic Pools. For dynamic pools, the PoolType is set to the dynamic value (65536) and the DynamicPoolType is set to an integer value used to distinguish the various dynamic pool types.
DynamicPoolType	This attribute holds the dynamic type value assigned to a dynamic memory pool. It is only valid when the PoolType attribute has the value Dynamic (65536).
AlternatePoolType	Indicates whether this MemoryPool has an alternate pool configured. Alternate pools are used for fallback when the current pool runs out of memory. If the value is set to zero, then this pool does not have an alternate. Otherwise the value is the same as the value of PoolType of the alternate pool.
IsValid	Indicates whether the other attributes in this MemoryPool contain accurate data. If an instance of this object has the value, False, (indicating an internal error condition), the values of the remaining objects in the instance may contain inaccurate information. That is, the reported values may be less than the actual values.
Used	Indicates the number of bytes from the MemoryPool that are currently in use by applications on the managed device.
Allocated	Indicates the number of bytes from the MemoryPool that are currently unused on the managed device.
Free	Indicates the largest number of contiguous bytes from the MemoryPool that are currently unused on the managed device.

OSElement (Cisco_OSElement)

OSElement represents the software element that is deployed to a network system and describes the software behind the operating system.The table below describes the other elements in OSElement.

Element	Description
InstanceName	Name of the OS image.
Family	Family of the OS component.
Version	Version of the OS.
Description	Description of the OS image.

IPProtocolEndpoint (Cisco_IPProtocolEndpoint)

IPProtocolEndpoint contains the subnet mask and default gateway information corresponding to the management IP Address.

Element	Description
Address	IP address of this endpoint, formatted according to the convention as defined in the AddressType property of this class.
SubnetMask	Mask for the IP address of this element, formatted according to the convention as defined in the AddressType property of this class (e.g., 255.255.252.0).
DefaultGateway	Default gateway address.

IfEntry (Cisco_IfEntry)

IfEntry represents the contents of an entry in the ifTable as defined in RFC 1213.

Element	Description
InstanceID	Index in the interface table defined in RFC 1213 for the entry containing the interface attributes of this object.
InstanceName	ifName attribute in the interface table defined in RFC 1213.
IfType	Interface type enumeration taken from the IANA definition of ifType.
IfSpeed	Estimate of the current bandwidth of the interface in bits per second. In cases, where the current bandwidth cannot be given, the nominal bandwidth should be specified.
IfAdminStatus	Desired state of the interface.
IfOperStatus	Current operational status of the interface.
Description	Description of the interface.
PhysicalAddress	Hardware address of the interface.
NetworkAddress	Network address of the interface.

Additional Information

AdditionalInformation is used to describe device specific information. It contains name and value attributes for elements specific to the device.

Class	Element	AdditionalInformation Tag
Cisco Call Manager	Cisco_NetworkElement	ActivePhones, InActivePhones, ActiveGateways, InActiveGateways, CallManagerIndex, CallManagerName, CallManagerDescription, CallManagerVersion, CallManagerStatus
	Cisco_Chassis	ApplicationPackageIndex, PackageManufacturer, Productname, Packageversion, Package SerialNumber

Class	Element	AdditionalInformation Tag
Cisco FastSwitch With_Module	Cisco_NetworkElement	FwdEngRev, BoardRev, SwitchPortNumber , SharedPortNumber, FirmwareSource
	Cisco_FlashDevice	FlashBankStatus
	Cisco_Card	InstanceID, ID
Cisco Firewall	None	None
Cisco IPX-IGX-BPX	Cisco_Chassis	InstanceName , Number
	Cisco_Card	SecondarySwRev, slotIndex, RAMId, ROMId, BRAMId, BOOTId, LocationWithinContainer, SecondaryStatus
	Cisco_Port	switchIfSlot,switchIfPort, SubPortNo, Status, Speed, PortType
Cisco LS1010 Switch	Cisco_Chassis	Slot0 (Type), Slot1(Type), AvailableSlots
	Cisco_NetworkElement	ConfigReg
	Cisco_PhysicalMemory	NVRAMUsed, RomVersion
Cisco MGX	Cisco_Chassis	Name, switchIfSlot, switchIfPort, SubPortNo, Status, Speed, PortType
Cisco Catalyst 3900 Switch	Cisco_Chassis	ModuleCount, Configuration, SwitchCount
	Cisco_Card	CiscoTsNumber, PermanentMAC, LocalMAC, CiscoTsModNumber , StackNo
Cisco Router 700 Series	Cisco_Chassis	MACAddress, PortCount, Type
Cisco Router	Cisco_PhysicalMemory	NVRAMUsed, ROMVersion
	Cisco_NetworkElement	Config
Cisco Catalyst IOS Switch	Cisco_Chassis	MACAddress, PortCount, Type
	Cisco_Card	FlashSize,FlashFree,FlashCard
	Cisco_Chassis	Config
	Cisco_PhysicalMemory	NVRAMUsed, ROMVersion
Cisco Catalyst L2L3 Switch	Cisco_Chassis	Slot0, Slot1 , MACAddress, PortCount, Type
	Cisco_NetworkElement	Config
	Cisco_PhysicalMemory	NVRAMUsed, ROMVersion
Cisco VPN 3000 Concentrators	Cisco_Chassis	PowerSupply1Type, PowerSuppy2Type, RAMSize
	Cisco_Card	LocationWithinContainer, CryptoHardwareType, SepState, DSPCodeVersion
Cisco Catalyst Switch	Cisco_Chassis	PowerSupply1, PowerSupply2 , MgmtType, BroadcastAddress, AvailableSlots
	Cisco_Card	ModuleIndex, RedundantModule, ModuleSubType
	Cisco_LogicalModule	moduleIndex,ModuleIPAddress
Cisco Optical Switches	Cisco_NetworkElement	RFUnitDetected, RFUnitID, RFUnitState, RFPeerUnitID, RFPeerUnitState, ActivateRF, ManualSwitchPermitted , StartupSyncStatus, RunningSyncStatus
	Cisco_PhysicalMemory	NVRAMUsed

Class	Element	AdditionalInformation Tag
Cisco FastSwitch	Cisco_NetworkElement	FwdEngRev, BoardRev, SwitchPortNumber , SharedPortNumber, FirmwareSource
	Cisco_FlashDevice	FlashBankStatus
	Cisco_Chassis	EPROMRev
	Cisco_CommunicationConnector	swPortIndex , PortTableSize, RevName, Type
Cisco Content Service Switch	None	None
Cisco Aironet	Cisco_PhysicalMemory	NVRAMUsed, ROMVersion
	Cisco_NetworkElement	Config
Cisco NAM	None	None
Cisco Management Engines	None	None

Overview: cwcli inventory Command

The `cwcli inventory` is a Device Management application command line tool. This tool supports the following features:

- You can check the specified device credentials for the devices.
- You can export device credentials of one or more devices in clear text.
- You can delete the specified devices.
- You can view the devices state.

The `cwcli inventory` command is located in the following directories, where *install_dir* is the directory in which LMS is installed:

- On Solaris and Soft Appliance systems, /opt/CSCOpx/bin
- On Windows systems, *install_dir*\CSCOpx\bin

  The default install directory is C:\Program Files.

This section contains:

- Using the cwcli inventory Command
- Running the cwcli inventory cda Command
- Running the cwcli inventory crmexport Command
- Running the cwcli inventory deletedevice Command
- Running the cwcli inventory getdevicestate Command

If you install LMS on an NTFS partition on Windows, only users in the administrator or casuser group can access `cwcli inventory`.

You can also perform the `cwcli inventory` tasks using the servlet. You will have to upload a payload XML file, which contains the `cwcli inventory` command arguments and LMS user credentials.

You have to write your own script to invoke the servlet with a payload of this XML file and the servlet returns the output either on the console or in the specified output file, if the credentials are correct and arguments are valid.

The name of the servlet is /rme/cwcli.

The following is the servlet to be invoked to run any command:

For post request,

perl *samplepost.pl* **http://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

perl *samplepost.pl* **https://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTPS mode is 443.

The schema for creating the payload file in XML format is:

```
<payload>
<command>
cwcli inventory commandname -u user -p BAse64 encoded pwd -args1 arg1value...
</command>
</payload>
```

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

The script and the payload file should be residing in the client machine.

For get request,

http://*lms-server:lms-port***/rme/cwcli?command=cwcli inventory** *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

https://*lms-server:lms-port***/rme/cwcli?command=cwcli inventory** *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTPS mode is 443.

The BAse64 encoded for “admin” is YWRtaW4=.

The URL encode for,

- Double quotes (“) is %22
- Percentage sign (%) is %25

Using the cwcli inventory Command

The command line syntax of the application is in the following format:

cwcli inventory *command GlobalArguments AppSpecificArguments*

The command line syntax of the application is in the following format:

cwcli export *command GlobalArguments AppSpecificArguments*

- **cwcli inventory** is the CiscoWorks command line interface for:
  - Checking the specified device credentials for the LMS devices.
  - Exporting device credentials of one or more LMS devices in clear text.
  - Deleting the specified LMS devices.

  - Viewing the LMS devices state.

- *Command* specifies which core operation is to be performed.
- *GlobalArguments* are the additional parameters required for each core command.
- *AppSpecificArguments* are the optional parameters, which modify the behavior of the specific `cwcli inventory` core command.

The order of the arguments and arguments are not important. However, you must enter the core command immediately after `cwcli inventory`.

The following sections describe:

- The `cwcli inventory` commands (See cwcli inventory Commands)
- The mandatory and optional arguments (See cwcli inventory Global Arguments)

On UNIX, you can view the `cwcli inventory` man pages by setting the MANPATH to /opt/CSCOpx/man. The man pages to launch the `cwcli inventory` are:

- man cwinventory-cda to launch the `cwcli inventory cda` command.
- man cwinventory-`delete` to launch the `cwcli inventory delete` command.
- man cwinventory-export to launch the `cwcli inventory export` command.
- man cwinventory-state to launch the `cwcli inventory getdevicestate` command

cwcli inventory Commands

The following table lists the command part of the `cwcli inventory` syntax:

Command	Description
`cwcli inventory cda`	You can check the specified device credentials for the devices. See Running the cwcli inventory cda Command
`cwcli inventory crmexport`	You can export device credentials of one or more devices in clear text. See Running the cwcli inventory crmexport Command
`cwcli inventory deletedevice`	You can delete the specified devices. See Running the cwcli inventory deletedevice Command
`cwcli inventory getdevicestate`	You can view the devices state. See Running the cwcli inventory getdevicestate Command

cwcli inventory Global Arguments

The following describes the mandatory and optional global arguments for `cwcli inventory`:

Argument	Description	Usage Notes
`-u` *user*	Enter a valid LMS username.	Mandatory.
`-p` *password*	Enter the password for the username.	Mandatory. You can provide this as part of argument or you can enter the password when you get the password prompt. You can also specify the password in a file. See Setting CWCLIFILE Environment Variable for more details.
{`-device` *name* | `-view` *name* | `-device` *list* `-view` *name*| `-ipaddress` *list*}	`device` *name*—Enter the Device Name of the device that you have added in the Device and Credentials database (Inventory > Device Administration > Add / Import / Manage Devices)	Mandatory • You can enter multiple device list separated using a comma. For example, if there are two devices with Device Names Rtr12 and Rtr13, using Rtr% will display both the devices. • To include all the devices, use the wild card character "%". For example, To use all the devices, use `-device` %.
	`view` *name*—Enter the Device Group name.	Mandatory You can enter multiple group name separated using a comma. For view name, you have to enter the fully qualified path as in the Group Administration GUI. For example, `-view` "/RME@*ciscoworks_servername*/ All Devices"
	`device` *list* `view` *name*—Enter the Device Name and the Device Group name.	Mandatory.
	`ipaddress` *list*—Enter the device IP4 address as entered in the Device and Credential Repository. You can enter multiple IP address with comma separated.	Mandatory. You cannot use this option with `-device`, `-view`, or `-input`. Also, you cannot specify wildcard characters.
[`-d` *debug_level*]	Enter the debug level.	Optional. *debug_level* is a number between 1 (the least information is sent to the debug output) and 5 (the most information is sent to the debug output). If you do not specify this argument, 4(INFO) is the default debug level.

Argument	Description	Usage Notes
[`-m` *email*]	Specify an e-mail address to send the results.	Optional. *email* is one or more e-mail addresses for notification. They can be separated by a space or comma.
[`-l` *logfile*]	Specify a file to which this command has to write log messages. The default log filename is cli.log. The default log directory is: On Windows: *NMSROOT*\log Where *NMSROOT* is the LMS installed directory. On Solaris and Soft Appliance: /var/adm/CSCOpx/log	Optional. Use the relative pathname to specify the *log_filename*.
`-help`	Displays command usage information	None.

Running the cwcli inventory cda Command

You can use this command to check the following device credentials:

- SNMP Read Community String—SNMP version 2 read community string.
- SNMP Write Community String—SNMP version 2 write community string.
- SNMP Version 3—SNMP version 3 username and password.
- Telnet—Telnet username and password.
- Telnet Enable Mode User Name and Password—Telnet username and password in Enable mode.
- SSH—SSH username and password.
- SSH Enable Mode User Name and Password—SSH username and password in Enable mode.

You can update these credentials using **Inventory > Device Administration > Add / Import / Manage Devices**.

The command syntax for **`cwcli inventory cda`** is:

`cwcli inventory cda -u` *userid* **`-p`** *password* {**`-invoke`** | **`-status`**} [**`-credType`** *credTypeL*ist] {**`-device`** *list* | **`-view`** *name* | **`-device`** *list* **`-view`** *name* | **`ipaddress`** *list*} [**`-d`** *debuglevel*] [**`-m`** *email*] [**`-help`**] [**`-l`** *logfile*]

Arguments in square brackets ([]) are optional; arguments in curly braces ({}) are required. You must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).

If you do not specify an optional argument, the default value configured for the system is used.

The following table describes the arguments that are specific to **`cwcli inventory cda`** command.

The other common arguments used by **`cwcli export`** are explained in Using the cwcli export Command

:

Argument	Description	Usage Notes
{`-invoke` | `-status`}	Invoke—Invokes the Check Device Attribute operation. Status—Displays the check device attributes result.	Mandatory. These arguments are mutually exclusive. You cannot run `-invoke` and `-status` together. After using the `-invoke` argument to the check device attribute you must run the command again with -status argument to view the result. If you are checking the device credentials for same devices and for same set of credentials, then you can use `-invoke` argument only once. If you are checking the device credentials for different devices and different credentials then you must use `-invoke` argument first and then you must use `-status`.
{`-credType` *credTypeList*}	Enter the device credentials for which you want to view the status. You can use the following arguments to view the different credentials status: • 0 —Enter **0** to view all credentials status. • 1 — Enter **1** to view status for Read Community. • 2 — Enter **2** to view status for Write Community. • 3 — Enter **3** to view status for SNMP version 3 username and password. • 4 — Enter **4** to view status for Telnet username and password. • 5 — Enter **5** to view status for Telnet username and password in Enable mode. • 6 — Enter **6** to view status for SSH username and password. • 7 — Enter **7** to view status for SSH username and password in Enable mode. You can specify multiple arguments separated by comma to check multiple credentials	Mandatory. If you do not specify the credentials type, all credentials status are displayed.

Argument	Description	Usage Notes
Command Argument for Inventory CDA `createjob`		
{**`-device`** *comma_separated_devicelist* }|{**`-view`** *device_view_name*}	**`-device`** *comma_separated_devicelist* - Specify devices to be used for the job. The *comma_separated_devicelist* is list of devices. **`-view`** *device_view_name* Specify the device view to be used for the job. The *device_view_name* is the name of a device view.	Mandatory. These arguments are mutually exclusive. You cannot run **`-device`** and **`-view`** together. **`cwcli inventory cda createjob -u`** *Username* **`-p`** *Password* **`-device`** *Device 1*, *Device 2* |**`-view`** *device_view_name*, **`-schedule`** *Schedule* **`-scheduletype`** *Schedule Type*
[{**`-schedule`** *MM/dd/yyyy:HH:mm:ss* **`-scheduletype`** *Once* | *Daily* | *Weekly* | *Monthly* | *LastDayOfMonth* | *6hourly* | *12hourly*}]	You can specify the date and time as well as the frequency of the CDA job. • To specify the date and time when you want to run the CDA job, use the **`schedule`** option. • To specify the frequency of the job use the **`scheduletype`** option. You have to set both the **`schedule`** and **`scheduletype`** options for a scheduled job. You do not have to set the **`schedule`** and **`scheduletype`** for an Immediate job.	Optional. *scheduletype* can have any of the following values: • Once • 6hourly • 12hourly • Daily • Weekly • Monthly If the **`schedule`** option is not specified, the job will be created as an immediate job.
[**`-input`** *argFile*]	Input file containing the details of the subcommands to be used for a job creation.	Optional If you are specifying the argument file, you need not specify the arguments in the command line. **`cwcli inventory cda -u`** *admin* **`-p`** *admin* [**`-input`** *argFile*]
[**`-description`** *JobDescription*]	Gives details of the job.	*JobDescription* is a user-defined entry describing the job details.
[**`-notificationmail`** *comma_separated_email_list*]	Specify the e-mail addresses to which the configuration job will sends status notices.	Optional Separate multiple addresses with commas.

Argument	Description	Usage Notes
{`-credType` *credentialList*}	Enter the device credentials for which you want to create a job. You can use the following arguments to view the different credentials status: • 0 —Enter **0** to view all credentials status. (ALL). • 1 — Enter **1** to view status for Read Community. • 2 — Enter **2** to view status for Write Community. • 3 — Enter **3** to view status for SNMP version 3 username and password. • 4 — Enter **4** to view status for Telnet username and password. • 5 — Enter **5** to view status for Telnet username and password in Enable mode. • 6 — Enter **6** to view status for SSH username and password. • 7 — Enter **7** to view status for SSH username and password in Enable mode. You can specify multiple arguments separated by comma to check multiple credentials.	Mandatory If you do not specify the credentials type, all credentials status are displayed.
Command Argument for Inventory CDA `stopjob`		
{`-id` *Job ID*}	You can stop only one job at a time. You can stop a CDA job that is in scheduled as well as running state.	Mandatory Use this command to stop an Inventory CDA job that is scheduled. `cwcli inventory cda stopjob -u` *userid* `-p` *password* {`-id` *jobID*}
Command Argument for Inventory CDA `deletejob` and `jobdetails`		
{`-id` *Job IDs*}	You can delete more than one job at a time. Enter the Job IDs that you want to delete, separated by commas. You can list the details of more than one job at a time. Enter the Job IDs separated by commas.	Mandatory Inventory CDA deletejob command: `cwcli inventory cda deletejob -u` *userid* `-p` *password* {`-id` *jobID1, jobID2..*} Inventory CDA jobdetails command: `cwcli inventory cda jobdetails -u` *userid* `-p` *password* {`-id` *jobID1, jobID2..*}

Argument	Description	Usage Notes
Command Argument for `Inventory CDA listjobs`		
[`-jobstatus` *status*]	You can specify the status of the job. This can be: • All jobs • Running jobs • Completed jobs • Pending jobs.	Optional. `cwcli inventory cda listjobs -u` *Username* `-p` *Password* [`-jobstatus` *A, R, C, P*]
Command Arguments for Inventory CDA `jobresults`		
{`-id` *Job ID, Job ID*}	You can list the results of more than one job at a time. Enter the job IDs separated by commas.	`cwcli inventory cda jobresults -u` *Username* `-p` *Password* `-id` *"Job ID 1" , "Job ID 2" , "Job ID 3"*
[`-csvoutput` *filepath*]	You can specify the fully qualified pathname for saving the job results.	If you do not specify this argument, the job results appear in the console itself. If the specified path does not exist, the job results are stored in the default location. `cwcli inventory cda jobresult -u` *admin* `-p` *admin* {`-jobid` *jobID*} [`-csvoutput` *filepath*]

The Table A-1 maps the device credentials that you have entered in the Device and Credentials (**Inventory > Device Administration > Add / Import / Manage Devices**) database and the credentials that appear in the `cwcli inventory cda` command:

Table A-1 Credentials Mapping

Credentials in Device and Credentials Database	Credentials displayed in `cwcli inventory cda`
Device Name	deviceName
SNMP V2C RO Community String	ro
SNMP V2C RW Community String	rw
SNMP V3 Username and Password	snmpv3
Primary Credentials Username	telnet
Primary Credentials Username and Primary Enable Password	telnetEnable
Primary Credentials Username	ssh
Primary Credentials Username and Primary Enable Password	sshEnable

Table A-2 describes the Credential Verification Report Status messages:

Table A-2 Understanding Credential Verification Report

Status Message	Description
OK	Check for device credentials completed. The device credentials data in the Device and Credential Repository matches the physical device credentials.
No authentication configured	Device was not configured with authentication mechanism (Telnet/LocalUsername/TACACS). LMS was able to use Telnet and log into the device successfully with out using the values entered in the Device and Credential Repository.
Incorrect	Check for device credentials completed. The device credentials data in the Device and Credential Repository does not match with the physical device credentials for one of the following reasons: • The device credentials data in Device and Credential Repository is not correct. • The device is unreachable or offline. • One of the interfaces on the device is down.
No Data Yet	Check for device credentials is in progress.
Did Not Try	Check for device credentials is not performed for one of the following reasons: • A Telnet password does not exist, so could not login to the device. • Device Telnet login mode failed, so enable mode login is not attempted.
No Value To Test	Check for device credentials is not performed because you have not entered the device credentials data.
Not Supported	Check for Telnet passwords is not performed because Telnet credential checking is not supported on this device.
Failed	Check failed because a Telnet session could not be established due to a not responding device.
Not Selected For Verification	Protocol was not selected for verification.

Usage Examples for cwcli inventory cda Command

This section provides some examples of usage for the **`cwcli inventory cda`** command.

Example 1: Invoking the Check Device Attributes

`cwcli inventory cda -u` *admin* **`-p`** *admin* **`-invoke -device`** 3750-stack

The command output is:

```
CDA invoked for given device and credType list
SUMMARY
========
        Successful: CDA: Success
```

Example 2: Checking the read and write device credentials for multiple devices

cwcli inventory cda -u admin **-p** admin **-device** 3750-stack, rtr% **-credtype** 1,2 **-status**

```
CDA Status :
============
deviceId | deviceName | ro | rw | snmpv3 | telnet | telnetEnable | ssh | sshEnable
25 | rtr10005 | OK | OK |  |  |  |  |
27 | 3750-stack | OK | OK |  |  |  |  |
32 | rtr10K | No Data Yet | No Data Yet |  |  |  |  |
SUMMARY
========
        Successful: CDA: Success
```

Example 3: Checking all device credentials for a group

cwcli inventory cda -u admin **-p** admin **-view** "/RME@*ciscoworksservername*/Pre-deployed" **-status**

```
CDA Status:
==========
deviceId | deviceName | ro | rw | snmpv3 | telnet | telnetEnable | ssh | sshEnable
29 | v2 | No Data Yet | No Data Yet | No Data Yet | No Data Yet | No Data Yet | No Data
Yet | No Data Yet
SUMMARY
========
        Successful: CDA: Success
```

Example 4: Checking device credentials for a device using the cwcli get request

The password that you enter here must be in base64 encoded. In this example, *YWRtaW4=* is the base64 encoded password for admin.

Enter this in your browser:

http://ciscowork_servername:1741/rme/cwcli?command=cwcli inventory cda -u *admin* **-p** *YWRtaW4=* **-device** *10.10.10.12* **-status**

The output for this appears on your console:

```
<!-- Processing Starts  -->
CDA Status :
============
deviceId | deviceName | ro | rw | snmpv3 | telnet | telnetEnable | ssh | sshEnable
12 | 10.10.10.12 | OK | OK | No Value To Test | Incorrect | Did Not Try | Failed | Did
Not Try
SUMMARY
========
Successful: CDA: Success
<!-- Processing complete -->
```

Example 5: Checking device credentials for a device using the cwcli post request

The password that you enter here must be in base64 encoded. In this example, YWRtaW4= is the base64 encoded password for admin.

The payload file, *cda.xml* contains:

```
<payload>
   <command>
     cwcli inventory cda -u admin -p YWRtaW4=  -device 10.10.16.20 -status
   </command>
</payload>
```

At the command prompt enter:

perl *samplepost.pl* **http://ciscowork_servername:1741/rme/cwcli** *cda.xml*

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

```
<!-- Processing Starts  -->
CDA Status :
============
deviceId | deviceName | ro | rw | snmpv3 | telnet | telnetEnable | ssh | sshEnab
le
71 | 10.10.16.20 | No Data Yet | No Data Yet | No Data Yet | No Data Yet | No
Data Yet | No Data Yet | No Data Yet

SUMMARY
========
        Successful: CDA: Success
<!-- Processing complete -->
```

Example 6: Creating a job using cwcli inventory cda createjob command

```
cwcli inventory cda createjob -u admin -p admin - Cat6230, Cat4500 | -view myview
-schedule 03/23/2007:12:15:01 -scheduletype Once
```

This command creates a cda job for the devices Cat6230 and Cat4500 in the view myview and scheduled for 23rd march 2007 at 12:15 pm with schedule type specified as Once.

Example 7: Stopping a cwcli inventory cda job using stopjob command

There is a job 1098, which is currently running. You can use this command to stop the job 1098.

```
cwcli inventory cda stopjob -u admin -p admin -id 1098
```

Example 8: Deleting cwcli inventory cda jobs using deletejob command

There are two jobs 1057 and 1058 scheduled. You can use this command to stop the two jobs.

```
cwcli inventory cda deletejob -u admin -p admin -id 1057,1058
```

Example 9: Getting details of jobs using cwcli inventory cda jobdetails command

There are two jobs 1001 and 1002 that are scheduled. You can use this command to list the details of both the jobs:

```
cwcli inventory cda jobdetails -u admin -p admin -id 1001, 1002
```

Example 10: Listing the cda jobs based on the status using the listjobs command

```
cwcli inventory cda listjobs -u admin -p admin -jobstatus R, C
```

Use this command to list those jobs whose status is Running or Completed.

Example 11: Obtaining results of jobs using jobresults comand

There are two jobs 1023 and 1024 that are completed. You can use this command to save the results of these jobs to the specified location.

```
cwcli inventory cda jobresult -u admin -p admin -jobid 1023, 1024 -csvoutput
C:/jobs/results
```

Running the cwcli inventory crmexport Command

You can use this command to export device credentials in CSV or XML format.

The command syntax for **cwcli inventory crmexport** is:

cwcli inventory crmexport -u *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] {**-device** *list* | **-view** *name* | **-device** *list* **-view** *name*} [**ipaddress** *list*] {**-filetype** *format* | **-filename** *outputfile*}

Arguments in square brackets ([]) are optional; arguments in curly braces ({}) are required. You must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).

If you do not specify an optional argument, the default value configured for the system is used.

The following table describes the arguments that are specific to **cwcli inventory crmexport** command. The other common arguments used by **cwcli export** are explained in Using the cwcli inventory Command.

Argument	Description	Usage Notes
{**-filetype** *format* }	**-filetype** *format* —Enter the file format to export, either XML or CSV.	Mandatory. The default CSV file format version is 3.0.
{ **-filename** *outputfile* }	**-filename** *outputfile*—Enter the filename.	Mandatory. Specifies the name of the file to which the device credentials information is to be exported on LMS server. If you are using **cwcli** remotely (get or post request), by default the output file is available at this location on LMS server: On Windows: *NMSROOT*\MDC\tomcat Where, *NMSROOT* is the LMS installed directory. On Solaris and Soft Appliance: *NMSROOT*/objects/dmgt

Usage Examples for cwcli inventory crmexport Command

This section provides some examples of usage for the **cwcli inventory crmexport** command.

Example 1: Exporting device credentials of all devices in XML format

cwcli inventory crmexport -device *%* **-filetype** *xml* **-filename** *crmexport-xml* **-u** *admin* **-p** *admin*

```
SUMMARY
======== Successful: Export: Success
```

The device credentials are exported into a file, *crmexport-xml* in XML format. The credentials that are exported depends on the data that you have provided when you added the devices to Device Credentials Repository.

Example 2: Exporting device credentials of all devices in Normal State in CSV format

cwcli inventory crmexport -view "*/RME@ciscoworksservername/Normal Devices*" **-filetype** *csv* **-filename** *crmexport-csv* **-u** *admin* **-p** *admin*

```
SUMMARY
========
        Successful: Export: Success
```

The device credentials for devices that are in Normal state are exported into a file, *crmexport-csv* in CSV version 3.0 format. The credentials that are exported depends on the data that you have provided when you added the devices to Device Credentials Repository.

Example 3: Exporting device credentials of all devices using cwcli get request method

The password that you enter here must be in base64 encoded.

In this example,

- *YWRtaW4=* is the base64 encoded password for admin.
- *%25* is the URL encode for "*%*"

Enter this in your browser:

http://*ciscowork_servername:***1741/rme/cwcli?command=cwcli inventory crmexport -u** *admin* **-p** *YWRtaW4=* **-device** *%25* **-filetype** *xml*
-filename *getxml*

The output is written in the *getxml* file. The *getxml* file is located:

On Windows:

NMSROOT\MDC\tomcat

Where, *NMSROOT* is the LMS installed directory.

On Solaris and Soft Appliance:

NMSROOT/objects/dmgt

Example 4: Exporting device credentials of all devices using cwcli post request method

The password that you enter here must be in base64 encoded. In this example, YWRtaW4= is the base64 encoded password for admin.

The payload file, *crmexport.xml* contains:

```
<payload>
   <command>
    cwcli inventory crmexport -u admin -p YWRtaW4=  -device 10.66.162.208 -filetype xml
-filename /opt/CSCOpx/crmexport-xml
```

```
   </command>
</payload>
```

At the command prompt enter:

perl *samplepost.pl* **http://ciscowork_servername:1741/rme/cwcli** *crmexport.xml*

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

```
SUMMARY
========
        Successful: Export: Success
```

The device credentials are exported into a file, *crmexport-xml* in XML format. This file is created in the /opt/CSCOpx directory. By default, the specified file is created in this location:

On Windows:

NMSROOT\MDC\tomcat

Where, *NMSROOT* is the LMS installed directory.

On Solaris and Soft Appliance:

NMSROOT/objects/dmgt

The credentials that are exported depends on the data that you have provided when you added the devices to Device Credentials Repository.

Running the cwcli inventory deletedevice Command

You can use this command to delete devices.

The device information will be retained in the Device Credentials Repository. This information will not be removed till you delete the device from Device Credentials Repository.

The command syntax for **cwcli inventory deletedevice** is:

cwcli inventory deletedevice -u *userid* **-p** *password* [**-d** *debuglevel*]
[**-m** *email*] [**-l** *logfile*] [**-view** *name*] {**-device** *list* | **-input** *inputfile* | **ipaddress** *list*}

Arguments in square brackets ([]) are optional; arguments in curly braces ({}) are required. You must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).

If you do not specify an optional argument, the default value configured for the system is used.

The following table describes the arguments that are specific to **cwcli inventory deletedevice** command. The other common arguments used by **cwcli export** are explained in Using the cwcli export Command.

Argument	Description	Usage Notes
-input *inputfile*	**-input** *inputfile*—Enter the full path of the file containing comma-separated list of device name as entered in Device Credentials Repository. The input file should be of this format: **-device** 1.1.1.1,2.2.2.2,3.3.3.3 or **-device** 1.1.1.1 **-device** 2.2.2.2 **-device** 3.3.3.3	Mandatory You must also enter the file format either CSV or txt.

Usage Examples for cwcli inventory deletedevice Command

This section provides some examples of usage for the **cwcli inventory deletedevice** command.

Example 1: To delete a device

cwcli inventory deletedevice -u *admin* **-p** *admin* **-device** *10.76.10.10*

```
<cwcli> INFO - Total number of devices deleted successfully: 1
SUMMARY
========
        Successful: Delete Device: Success
```

Example 2: To delete devices listed in a file

The input file, deletedevice contains list of device Device Name separated by a comma:

-device 3750-stack,rtr1000,rtr10005

cwcli inventory deletedevice -u *admin* **-p** *admin* **-input** *deletedevice.csv*

Example 3: To delete devices using cwcli get request

The password that you enter here must be in base64 encoded. In this example, *YWRtaW4=* is the base64 encoded password for admin.

Enter the following in your browser:

http://ciscowork_servername:1741/rme/cwcli?command=cwcli inventory deletedevice -u *admin* **-p** *YWRtaW4=* **-device** *10.10.10.41,10.10.10.51*

The output for this appears on your console:

```
<!-- Processing Starts  -->
<cwcli> INFO - Total number of devices deleted successfully: 2
SUMMARY
========
        Successful: Delete Device: Success
<!-- Processing complete -->
```

Example 4: To delete devices using cwcli post request

The password that you enter here must be in base64 encoded. In this example, *YWRtaW4=* is the base64 encoded password for admin.

The payload file, *deletedevicestate.xml* contains:

```
<payload>
   <command>
     cwcli inventory deletedevice -u admin -p YWRtaW4=  -device
10.77.9.10,10.77.9.18,10.76.8.6
   </command>
</payload>
```

At the command prompt enter:

perl *samplepost.pl* **http://doclab2:1741/rme/cwcli** *deletedevice.xml*

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

```
<!-- Processing Starts  -->
<cwcli> INFO - Total number of devices deleted successfully: 3


SUMMARY
========
        Successful: Delete Device: Success
<!-- Processing complete -->
```

Running the cwcli inventory getdevicestate Command

You can use this command to view the device state.

The command syntax for **cwcli inventory getdevicestate** is:

cwcli inventory getdevicestate -u *userid* **-p** *password* [**-d** *debuglevel*] [**-m** *email*] [**-l** *logfile*] [**-view** *name*] {**-device** *list* | **-input** *inputfile* | **ipaddress** *list*}

Arguments in square brackets ([]) are optional; arguments in curly braces ({}) are required. You must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).

If you do not specify an optional argument, the default value configured for the system is used.

The following table describes the arguments that are specific to **cwcli inventory getdevicestate** command. The other common arguments used by **cwcli export** are explained in Using the cwcli inventory Command.

Argument	Description	Usage Notes
-input *inputfile*	**-input** *inputfile*—Enter the full path of the file containing comma-separated list of devices name as entered in Device Credentials Repository. The input file should be of this format: **-device** 1.1.1.1,2.2.2.2,3.3.3.3 or **-device** 1.1.1.1 **-device** 2.2.2.2 **-device** 3.3.3.3	Mandatory You must also enter the file format either CSV or txt.

Usage Examples for cwcli inventory getdevicestate Command

This section provides some examples of usage for the **cwcli inventory getdevicestate** command.

Example 1: To view the device state of the devices

cwcli inventory getdevicestate -u *admin* **-p** *admin* **-device** *10.10.19.10,10.10.19.12*

```
<cwcli> INFO - Device State Information
DisplayName:Device State
10.10.19.10:PREDEPLOYED
10.10.19.12:NORMAL
SUMMARY
========
        Successful: getdevicestate: Success
```

Example 2: To view the devices state specified in a file

The input file, deletedevice contains list of device name separated by a comma:

-device VG200,rtr1750,cat4000

cwcli inventory deletedevice -u *admin* **-p** *admin* **-input** *devicestate.csv*

Example 3: To view the devices state using get request

The password that you enter here must be in base64 encoded. In this example, *YWRtaW4=* is the base64 encoded password for admin.

Enter the following in your browser:

http://ciscowork_servername:1741/rme/cwcli?command=cwcli inventory getdevicestate -u *admin* **-p** *YWRtaW4=* **-device** *10.16.10.15,10.16.10.35*

The output for this appears on your console:

```
<!-- Processing Starts  -->
<cwcli> INFO - Device State Information
DisplayName:Device State
```

```
10.16.10.15:NORMAL
10.16.10.35:PREDEPLOYED

SUMMARY
========
        Successful: getdevicestate: Success
<!-- Processing complete -->.
```

Example 4: To view the device state using post request

The password that you enter here must be in base64 encoded. In this example, YWRtaW4= is the base64 encoded password for admin.

The payload file, *getdevicestate.xml* contains:

```
<payload>
   <command>
     cwcli inventory getdevicestate -u admin -p YWRtaW4=  -device
12.20.12.26,10.6.12.21,12.18.10.129,10.7.9.13
   </command>
</payload>
```

At the command prompt enter:

`perl` *samplepost.pl* **`http://ciscowork_servername:1741/rme/cwcli`** *getdevicestate.xml*

To invoke the servlet using a script, see the Sample Script to Invoke the Servlet.

```
<!-- Processing Starts  -->
<cwcli> INFO - Device State Information
DisplayName:Device State
12.18.13.129:ALIAS
10.7.9.13:PREDEPLOYED
10.6.12.21:NORMAL
12.20.12.26:NORMAL

SUMMARY
========
        Successful: getdevicestate: Success
<!-- Processing complete -->
```

Overview: cwcli invreport Command

The **`cwcli invreport`** is a CiscoWorks command line tool which allows you to run previously created Inventory Custom Reports and also system reports. The supported output file format is Comma Separated Value (CSV).

The above command retrieves the inventory report in CSV format. The **`-file`** parameter stores the output in the file specified by *filename*. If you have not specified the filename, the output is stored at the following location:

- *NMSROOT*\files\rme\cri\archives\inventory*reportname_timestamp*.csv (On Windows)

- /var/adm/CSCOpx/files/rme/cri/archives/inventory/*reportname_timestamp.csv* (On Solaris and Soft Appliance)

NMSROOT is the LMS install directory.

You can:

- Use the `-reportname` argument to generate the report.

  This can be the name of:

  – An already defined custom template

  or

  – A system report name such as Detailed Device Report.

- Use the `-input` argument to specify a file containing the parameters for the report generation.

Note The `-view` argument is not allowed in the input file.

- Enable debug mode and set the debug level using the `-d` argument.
- E-mail the output to an e-mail recipient using the `-m` argument.
- Log the error messages to a file using the `-l` argument. The log and the output files are created in the current directory.
- List the existing reports with the `-listreports` argument.

Running the cwcli invreport Command

To use the `cwcli invreport` command, you must be able to run the cwcli command

You should be authorized to generate inventory reports.

The command syntax is:

`cwcli invreport -u` *userid* `-p` *password* [`-d` *debuglevel*] [`-m` *email*] [`-l` *logfile*] {`-listreports` | `-reportname` *name* {`-view` *viewname* | `-device` *list* | `-ipaddress` *list*} [`-file` *filename*] | `-input` *inputfile*}

Arguments in square brackets ([]) are optional; arguments in curly braces ({}) are required. You must provide one argument from each group of arguments in curly braces ({}) that is separated by vertical bars (|).

If you do not specify an optional argument, the default value configured for the system is used. Valid values for arguments are described in the following table:

Argument	Description	Usage Notes
`-u` *user*	Provide valid LMS username.	None.
`-p` *password*	Provide password for username. You can also specify the password in a file. See Setting CWCLIFILE Environment Variable for more details.	None.

Argument	Description	Usage Notes
[`-d` *debug_level*]	Set debug level.	Optional. *debug_level* is a number between 1 (the least information is sent to the debug output) and 5 (the most information is sent to the debug output). If you do not specify this argument, 4(INFO) is the default debug level.
[`-m` *email*]	Specify an e-mail address to send the results.	Optional. *email* is one or more e-mail addresses for notification. They can be separated by a space or comma.
[`-l` *log_filename*]	Logs the error messages and debug of messages of the invreport command, to the specified logfile name. If not specified, it will be written to default logs (invreports.log and cli.log).	Optional. *log_filename* can be full pathname or filename in local directory.
[`-l` *log_filename*]	Logs the error messages and debug of messages of the invreport command, to the specified logfile name. If not specified, it will be written to default logs (invreports.log and cli.log).	Optional. *log_filename* can be full pathname or filename in local directory.

Argument	Description	Usage Notes
{`-listreports` | `-reportname` *name* {`-view` *viewname* | `-device` *list* | `-ipaddress` *list*} [`-file` *filename*] | `-input` *inputfile*}	Specify any one of the required arguments: • `-listreports` • `-reportname` • `-input`	`-listreports` argument lists out all Inventory system reports and custom reports templates. You can run this command if you have the required permissions to generate reports. `-reportname` *name* specifies the name of an already defined custom template or the name of a system report (such as Detailed Device Report) for which the CSV formatted report is to be generated. `-input` *inputfile* specifies the input file containing report parameters. The parameters in this file will be used to generate the CSV formatted report(s). For Solaris and Soft Appliance, you must specify the complete path of the input file. For Windows, this file should be located in the current directory or you can specify the complete path of the input file. The `-view` argument is not allowed in the input file.

Argument	Description	Usage Notes
	If you selected `-reportname` *name,* then specify any one of these arguments: – `-view` *viewname*. This confines the device search to the specified view. – `-device` *list.* This specifies one or more device names as a comma-separated list. Optionally, you can also specify `-file` *filename.* Name of the file where CSV formatted report will be stored. If you do not specify the location, the default location is: *NMSROOT*\files\rme\cri\archives\inventory*reportname_timestamp.csv* (On Windows) /var/adm/CSCOpx/files/rme/cri/archives/inventory/*reportname_timestamp.csv* (On Solaris and Soft Appliance) – `ipaddress` *list*—This specifies IP4 address as entered in the Device and Credential Repository. You can enter multiple IP address with comma separated. You cannot use this option with `-device`, `-view` , or `-input`. Also, you cannot specify wildcard characters.	

Usage Examples

This section provides some examples of usage for the cwcli invreport command:

- Example 1
- Example 2
- Example 3
- Example 4
- Example 5
- Example 6

Example 1

`cwcli invreport -u` *admin* `-p` *admin* `-reportname` *"Detailed Device Report"* `-device` *%*

This generates Detailed Device Report for all devices and CSV file will be located at

- *NMSROOT*\files\rme\cri\archives\inventory*Detailed Device Report_timestamp.csv* (On Windows)
- /var/adm/CSCOpx/files/rme/cri/archives/inventory/*Detailed Device Report_timestamp.csv.* (On Solaris and Soft Appliance)

Example 2

cwcli invreport -u *admin* **-p** *admin* **-reportname** *"Detailed Device Report"* **-device** % **-file** *D:\cisco\CSCOpx\a.csv*

This generates Detailed Device Report, a system report, for all devices, and the result will be written to D:\cisco\CSCOpx\a.csv

Example 3

cwcli invreport -u *admin* **-p** *admin* **-reportname** *"Detailed Device Report"* **-device** % **-file** a.csv

This generates the Detailed Device Report, a system report, for all devices, and the result will be written to the file a.csv in the current directory (from where you are running this command).

Example 4

cwcli invreport -u *admin* **-p** *admin* **-input** *cliinputs.txt*

Generate the reports using the parameters provided in the file cliinputs.txt. Using **-input** argument you can run multiple reports at a time by providing parameters in the file.

Example 5

cwcli invreport -u *admin* **-p** *admin* **-listreports**

Displays a list of all Inventory system report and custom templates.

You can run this command if you have the required permissions to generate reports.

Example 6

cwcli invcreport -u *admin* **-p** *admin* **-d** 3 **-m** *xxx@yyy.com* **-reportname** *acmeinventory* **-view** acme **-file** *acmeinventory.txt*

Generates the report named acmeinventory, using the acme device view and the CSV formatted output will be written to acmeinventory.txt

You can place this file in the current directory (from where you are running the command).

Example 7

cwcli invreport -u *admin* **-p** *admin* **-reportname** *HardwareStatisticsReport* **-device** *devname* **-file** *hwstreport.txt*

Generates the Hardware Statistics Report for the device devname and the CSV formatted output will be written to hwstreport.txt

Example 8

cwcli invreport -u *admin* **-p** *admin* **-reportname** *DeviceStatisticsReport* **-device** *devname* **-file** *devstreport.txt*

Generates the Device Statistics Report for the device devname and the CSV formatted output will be written to devstreport.txt

Example 9

cwcli invreport -u *admin* **-p** *admin* **-reportname** *POEReport* **-device** *devname* **-file** *report.txt*

Generates the POE Report for the device devname and the CSV formatted output will be written to report.txt

cwcli invreport Remote Access

You can also perform the **cwcli invreport** tasks using the servlet. You will have to upload a payload XML file, which contains the **cwcli invreport** command arguments and LMS user credentials.

You have to write your own script to invoke the servlet with a payload of this XML file and the servlet returns the output either on the console or in the specified output file, if the credentials are correct and arguments are valid.

The name of the servlet is /rme/cwcli.

The following is the servlet to be invoked to execute any command:

For post request,

perl *samplepost.pl* **http://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

perl *samplepost.pl* **https://***lms-server:lms-port***/rme/cwcli** *payload_XML_file*

The default port for LMS server in HTTPS mode is 443.

The schema for creating the payload file in XML format is:

```
<payload>
<command>
cwcli inventory commandname -u user -p BAse64 encoded pwd -args1 arg1value...
</command>
</payload>
```

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

The script and the payload file should be residing in the client machine.

For get request,

http://*lms-server:rme-port***/rme/cwcli?command=cwcli invreport** *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTP mode is 1741.

If you have enabled SSL on LMS server, you can also use https protocol for secured connection.

https://*lms-server:lms-port***/rme/cwcli?command=cwcli invreport** *commandname* **-u** *user* **-p** *BAse64 encoded pwd -args1 arg1value...*

The default port for LMS server in HTTPS mode is 443.

The BAse64 encoded for “admin” is YWRtaW4=.

The URL encode for,

- Double quotes (“) is %22
- Percentage sign (%) is %25

Overview: cwcli netshow Command

You can invoke NetShow features from Command Line Interface (CLI).

The **cwcli netshow** commands let you use NetShow features from the command line. You can use the **cwcli netshow** commands to view, browse, create, delete, and cancel NetShow jobs.

You can also view the Command Sets assigned to each user by entering the command **listcmdsets** from CLI.

You can set the following job attributes using the command line option:

- E-mail Notification
- Job Based Password
- Execution Policy
- Approver List

However, the Administrator must define and assign the command sets to you, in the browser interface.

If you do not have permission to run custom commands, you can run a command or command set from the CLI only if:

- The command set is assigned to you by the Administrator.
- The command set has at least one command that can be run on the specified device.

If you have permission to run custom commands, you can run any of the following adhoc commands:

- **show**
- **version**
- **where**
- **ping**
- **traceroute**
- **?**

Administrator level users have all command sets assigned to them. However, only system-defined command sets are assigned to all users, by default. Other commands have to be assigned to the user by the Administrator. If any users create a command, it is automatically assigned to them.

Running cwcli netshow Command

The command syntax for running **cwcli netshow** commands is:

cwcli netshow *common_arguments subcommands command_arguments*

In the CLI version, you can provide the arguments in the (operating system shell) command line or in an input file. The input file provides you with flexibility and control over commands and command sets. You can specify the devices on which you want to run the command sets.

In the input file, you can include subcommands and command arguments.

For example, you can create a new netshow job with command sets, set1 and set2, and the custom commands, custom command 1 and custom command 2, by entering:

cwcli netshow createjob -u *Username* **-p** *Password* **-commandset** "*Command Set 1*"," *Command Set 2*" **-device** *Device 1*, *Device 2* **-customcmd** "*Custom command 1*"," *Custom command 2*" **-schedule** *Schedule* **-scheduletype** *Schedule Type*

Items in square brackets ([]) are optional; items in curly brackets ({}) are required.

The arguments are described in the following sections.

Subcommands

Subcommands specify the actions that you perform. Valid values for subcommands are described in the following table.

Sub Command	Description	Usage Notes	Example
createjob	Creates a new job that can be scheduled to run immediately or to be run sometime in the future. You can also specify the job attributes you want to enable.	Either use an input file containing the details of the subcommands or enter the full command syntax.	**cwcli netshow createjob -u** *Username* **-p** *Password* **-commandset** "*Command Set 1*"," *Command Set 2*" **-device** "*Device Name 1*", "*Device Name 2*" **-customcmd** "*Custom Command 1*", "*Custom Command 2*" **-schedule** *Schedule* **-scheduletype** *Schedule Type* Or **cwcli netshow createjob -u** *Username* **-p** *Password* **-input** *Input File*
canceljob	Cancels an existing job.	Enter the job ID.	**cwcli netshow canceljob -u** *Username* **-p** *Password* **-id** *"Job ID"*
deletejob	Deletes existing jobs.	Enter the job IDs separated by commas.	**cwcli netshow deletejob -u** *Username* **-p** *Password* **-id** "*Job ID 1*"," *Job ID 2*"
jobdetails	Displays details of specified job.	Enter the job IDs separated by commas.	**cwcli netshow jobdetails -u** *Username* **-p** *Password* **-id** "*Job ID 1*","*Job ID 2*", "*Job ID 3*"

`listjobs`	Displays a list of jobs created by the user and the job status.	Specify the type of jobs to be listed. The command type is case sensitive. The commands that you can use are: `A` —All jobs `R` —Running jobs `C` —Completed jobs `P` —Pending jobs You can use combinations of status options. Separate the options by commas.	`cwcli netshow listjobs -u` *Username* `-p` *Password* `-status` *R,C*
`listcmdsets`	Displays a list of command sets assigned to the user.	None.	`cwcli netshow listcmdsets -u` *Username* `-p` *Password*
`jobresults`	Displays results of specified jobs	Specify the job IDs. Separate the job IDs by commas.	`cwcli netshow jobresults -u` *Username* `-p` *Password* `-id` *"Job ID 1" , "Job ID 2" , "Job ID 3"*
`help`	Displays command usage information.	None.	`cwcli netshow -help`

Common Arguments

Common arguments specify parameters that apply to all subcommands. Valid values for *common_arguments* are described in the following table.

Command	Description	Usage Notes
`-u` *user*	Enter a valid LMS username.	None
`-p` *password*	Enter the password for the username. You can also specify the password in a file. See Setting CWCLIFILE Environment Variable for more details.	None
`[-d` *debug_level*`]`	Set the debug level.	Optional *debug_level* should be a number between 1-5. 1 —The least information is sent to the debug output 5 —The most information is sent to the debug output.
`[-l` *log_filename*`]`	Identifies a file to which Network Show Commands will write log messages. If you do not specify this, the log output will appear on screen.	Optional *log_filename* can be a full path to the file or a filename in the local directory.
`[-m` *Email ID*`]`	Enter your Email ID	You will get the output of the CLI operation in an Email.

Command Arguments

Command arguments specify parameters that apply only to specific subcommands. Valid values for command arguments are described in the following table.

Arguments in square brackets ([]) are optional. Arguments in curly brackets ({}) are required. You must provide one argument from each group of arguments in curly brackets ({}) that is separated by vertical bars (|).

Command Arguments	Description	Usage Notes
Command Arguments for `createjob`		
{**`-device`** *devicelist* | **`-view`** *view_name*}	Defines devices on which you want the command set to run.	*device_list* —List of device names. Separate these names by commas. *view_name* — Name of a device view.
{**`-commandset`** *commandset*}	Defines available command sets that you want to run on the selected devices.	*commandset* is the name of the command set that was assigned to you. You can specify more than one command set separated by commas. The command set name is case sensitive. You must specify *command set* or *custom command* or both to create a job.
{**`-customcmd`** *customcommand*}	Defines the user-defined commands that you want to run on the selected devices.	*customcommand* is a user-defined show command. You must specify *command set* or *custom command* or both to create a job. The custom commands which can be run on NetShow are: • **`show`** • **`version`** • **`where`** • **`ping`** • **`traceroute`** • **`?`** You can use the short forms of these commands. For example, **`sh`** for **`show`**.
[**`-description`** *description*]	Gives details of the job.	*description* is a user-defined entry describing the job details.

Command Arguments	Description	Usage Notes
[{**-schedule** *MM/dd/yyyy:HH:mm:ss* **-scheduletype** *Once* | *Daily* | *Weekly* | *Monthly* | *LastDayOfMonth* | *6hourly* | *12hourly*}]	You can specify the date and time as well as the frequency of the NetShow job. • To specify the date and time when you want to run the NetShow job, use the schedule option. • To specify the frequency of the job use the scheduletype option. You have to set both the schedule and schedule type options for a scheduled job. You do not have to set the schedule and schedule type for an Immediate job.	*scheduletype* can have any of the following values: • Once • 6hourly • 12hourly • Daily • Weekly • Monthly • LastDayOfMonth If the schedule option is not specified, the job will be created as an immediate job.
[**-makercomments** *comments*]	Job creator's comments to Job approver.	
[**-mkemail** *email*]	Maker e-mail ID for sending approval notifications	
[**-notificationmail** *email]*	Defines the e-mail addresses of persons who need to get mails when the job has started and completed.	*email* can contain a comma separated email list. If you do not specify this option in the CLI, the e-mail address specified in the UI are used.
[**-execution** *Sequential|Parallel*]	Execution policy. Specifies the order in which you want to run the job on the devices. Parallel—Allows the job to run on multiple devices at the same time. By default, the job runs on five devices at a time. Sequential—Allows the job to run on only one device at a time.	If you do not specify these options in the CLI, the corresponding settings from the UI are used.
{**-primary_user** *username* **-primary_pass** *password*}	Primary username and password to connect to devices.	
{**-enable_pass** *password*}	Execution mode password to connect to device.	
[**-input** *input file*]	Input file containing the details of the subcommands	If you are specifying the input file, you do not need to specify the subcommands.

Command Arguments	Description	Usage Notes
Command Argument for `listjob`		
{`-status` *status*}	You can specify the status of the job. This can be: • All jobs • Running jobs • Completed jobs • Pending jobs.	
Command Argument for `canceljob`		
{`-id` *Job ID*}	You can cancel only one job at a time.	
Command Argument for `deletejob` and `jobdetails`		
{`-id` *Job ID, Job ID}*	You can delete more than one job at a time. Enter the Job IDs that you want to delete, separated by commas. You can list the details of more than one job at a time. Enter the Job IDs separated by commas.	
Command Arguments for `jobresults`		
{`-id` *Job ID, Job ID*}	You can list the results of more than one job at a time. Enter the job IDs separated by commas.	
[`-output` *file path*]	You can specify the fully qualified pathname for saving the job results.	If you do not specify this argument, the job results appear in the console itself. If the specified path does not exist, the job results are stored in the default location.

Executing Netshow CLI Remotely

You can run NetShow CLI from a remote console.

NetShow uses the Remote Access feature in the CLI framework to help you to invoke the NetShow commands from the client in the same way as you run them on the LMS server.

The name of the servlet, to be invoked, is /rme/cwcli.

You must invoke the following URLs to run any command.

- For POST request:

  `http://`*lms-server:lms-port*/rme/`cwcli` *payload XML file*

- For GET request:

  `http://`*lms-server:lms-port*/rme/cwcli?command=`cwcli netshow` *command* `-u` *Username* `-p` *Password command_specific_args*

The contents of the payload.xml is:

```
<payload>
   <command>
    cwcli netshow command -u Username -p Password command_specific_args
   </command>
</payload>
```

For example to run the **listcmdsets** command payload.xml will be as follows:

```
<payload>
<command>
cwcli netshow listcmdsets -u Username -p Password
</command>
</payload>
```

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

The script and the payload file should be residing in the client machine.

Performance Tuning Tool

Performance Tuning Tool (PTT) is a Command Line Interface (CLI) utility that enables you to apply and list various profiles available in LMS server. Profiles consists of configuration files, which are in the form of XML files whose values are based on the recommendations for various applications. For more information on PTT features, refer to PTT Features.

There are two profiles shipped with LMS. You can use any of the profile that matches the system. For more information on PTT Profiles, see Profiles and PTT.

There maybe multiple configuration files that are involved while applying a profile. The parameters such as, snmp.threads.min, snmp.threads.max, ICSThreadCount, ICS DBConnectionCount, ThreadPoolCount, CDLNumOfThreads, max_db_connections, max_threads_for_config_fetch, EssentialsDMServicesHeapsize,ConfigJobManager.heapsize, and CDA_MIN_THREADS are tuned and available in each profile. You can apply the required profile to the system and improve performance. This is a major advantage of using PTT.

To know more about the command usage, see PTT Commands.

PTT Features

The PTT CLI utility allows you to:

- List the profile that is currently applied to the target machine.
- List the profiles that match the system configuration.
- List the profiles that match the operating system.
- Apply a selected profile onto the target machine.
- Reverse the changes done to a target machine by applying the default profile to restore the default settings.
- View details of a profile.

PTT Commands

Table A-3 lists the various PTT command options that you can use. These command options are common for Windows and Solaris and Soft Appliance.

Table A-3 PTT Command Options

PTT command options	Description
`-apply <profileName>`	Applies a particular profile. To reset, specify the default profile name as parameter and apply that profile.
`-apply`	Finds the matching profile and applies it automatically in a single step.
`-apply Default`	Finds the default profile and applies it automatically in a single step.
`-show`	Displays the currently applied profile.
`-list`	Lists all the profiles that match the target operating system.
`-list Match`	Lists the profile that matches the system configuration.
`-view <profileName>`	Displays the profile details. The details of the profile, which is specified in the command is displayed.
`rmeptt -help`	Displays help for all commands.

Command Usage

In Windows enter:

`rmeptt.bat <option> <argument>`

For example, to list all the profiles that matches the target operating system, the command is:

`rmeptt.bat -list`

In Solaris and Soft Appliance, enter:

`rmeptt.sh <option> <argument>`

For example to display the profile details, the command is:

`rmeptt.sh -view x`

Where *x* is the name of the profile.

Profiles and PTT

Profiles are XML files whose values are based on the recommendations of the various LMS applications. Each profile (XML file) consists of tuned parameters which when applied, improves performance.

There are two profiles that are shipped with LMS. They are:

- Default Profile
- Perftune - Windows and Perftune - Solaris and Soft Appliance

Note All the configuration files are backed up before applying a profile.

PTT identifies the matching profile for a LMS server based on the following criteria:

- The operating system for which the profile is created.
- The System Configurations such as Dual CPU and 4 GB RAM.

A profile is considered matching only if it meets these criteria.

When you apply a profile, the tuned parameters, see Table A-4 corresponding to that profile is applied to the system.

These parameters belong to Sync Archive, Netconfig, Syslog, Device Management, Check Device Attributes (CDA) and Inventory Collection sub systems of the LMS application. The profile, with tuned parameters when applied, improves the performance.

Note Ensure that the Daemon manager is stopped, before running PTT.

Example 1

If the default profile is applied to a system which already has a Perftune - Windows profile applied to it, the parameters are rolled back to original values. See Table A-4 for Original values.

Example 2

If the Perftune - Windows profile is applied to a system which already has a default profile applied, the parameters are changed from the original values to new values. See Table A-5 for Original and New values.

Default Profile

A default profile is a profile with default values. It is used to rollback the changes done by PTT. You can roll back the changes made to a profile, by applying the default profile. This action rolls back the parameters to their original values. The parameters and the original values are:

Table A-4 Default Profile Original Values

Sub system	Parameters	Original Values	Platform Supported
CDA	CDA_MIN_THREADS	7	Windows and Solaris and Soft Appliance
EssentialsDM	ConfigJobManager.heapsize	192m	Windows and Solaris and Soft Appliance
EssentialsDM	EssentialsDMServiceHeapsize	256	Windows and Solaris and Soft Appliance
Inventory Collection	snmp.threads.min	10	Windows and Solaris and Soft Appliance
Inventory Collection	snmp.threads.max	15	Windows and Solaris and Soft Appliance
Inventory Collection	ICS ThreadCount	10	Windows and Solaris and Soft Appliance
Inventory Collection	ICS DBConnectionCount	5	Windows and Solaris and Soft Appliance
NetConfig and SyncArchive	max_threads_for_config_fetch	5	Windows and Solaris and Soft Appliance
NetConfig and SyncArchive	ThreadPoolCount	10	Windows and Solaris and Soft Appliance

Table A-4 Default Profile Original Values

Sub system	Parameters	Original Values	Platform Supported
NetConfig and SyncArchive	CDLNumOfThreads	5	Windows and Solaris and Soft Appliance
NetConfig and SyncArchive	max_db_connections	20	Windows and Solaris and Soft Appliance
Config Management (Config Management Server Daemons - dmgtd.conf Arguments for max heap size.)	Xmx	192	Solaris and Soft Appliance
Config Management (Config Management Server Daemons - dmgtd.conf Arguments for minimum heap size.)	Xms	64	Solaris and Soft Appliance

Perftune - Windows and Perftune - Solaris and Soft Appliance

This profile consists of parameters that are tuned to improve performance.

- Perftune - Windows profile is applied to a system that has a Windows operating system, provided the profile matches the required criteria.
- Perftune - Solaris and Soft Appliance profile is applied to a system that has a Solaris and Soft Appliance operating system, provided the profile matches the required criteria.

  See Profiles and PTT for more informationon criteria for a profile to match a system.

The parameters that can be tuned are:

Table A-5 Perftune - Windows and Perftune - Solaris and Soft Appliance Parameters

Sub system	Parameters	Original Values	New Value	Platform Supported
CDA	CDA_MIN_THREADS	7	14	Windows and Solaris and Soft Appliance
EssentialsDM	ConfigJobManager.heapsize	192m	256	Windows and Solaris and Soft Appliance
EssentialsDM	EssentialsDMServiceHeapsize	256	512	Windows and Solaris and Soft Appliance
Inventory Collection	snmp.threads.min	10	20	Windows and Solaris and Soft Appliance
Inventory Collection	snmp.threads.max	15	25	Windows and Solaris and Soft Appliance
Inventory Collection	ICS ThreadCount	10	20	Windows and Solaris and Soft Appliance
Inventory Collection	ICS DBConnectionCount	5	10	Windows and Solaris and Soft Appliance
NetConfig and SyncArchive	max_threads_for_config_fetch	5	10	Windows and Solaris and Soft Appliance

Table A-5 Perftune - Windows and Perftune - Solaris and Soft Appliance Parameters

Sub system	Parameters	Original Values	New Value	Platform Supported
NetConfig and SyncArchive	ThreadPoolCount	10	20	Windows and Solaris and Soft Appliance
NetConfig and SyncArchive	CDLNumOfThreads	5	20	Windows and Solaris and Soft Appliance
NetConfig and SyncArchive	max_db_connections	20	40	Windows and Solaris and Soft Appliance
Config Management (Config Management Server Daemons - dmgtd.conf Arguments for max heap size.)	Xmx	192	256	Solaris and Soft Appliance
Config Management (Config Management Server Daemons - dmgtd.conf Arguments for minimum heap size.)	Xms	64	128	Solaris and Soft Appliance

syslogConf.pl Utility

The syslogConf.pl is a perl script CLI utility. You can use this utility to:

- Change Syslog Analyzer Port.
- Change Syslog Collector Port.
- Configure Remote Syslog Collector (RSAC) Address and Port in LMS server.
- Change Syslog File Location.

You can run this script in the LMS server as well as the RSAC server. All the activities mentioned above can be performed in a LMS server by running the syslogConf.pl script from the command prompt.

In RSAC server, you can only change the Syslog Collector Port and Syslog File location. The Syslog Collector and Syslog Analyzer ports can be any number between 1025 and 5000.

This utility is available under:

NMSROOT/bin/ (On Solaris and Soft Appliance)

NMSROOT\bin (On Windows)

NMSROOT is the LMS install directory. For Solaris and Soft Appliance, it will be /opt/CSCOpx.

A log file for the syslogconf.pl script is available at:

In Solaris and Soft Appliance

/var/adm/CSCOpx/log/SyslogConf.log

In Windows

NMSROOT\log\SyslogConf.log

Note Before you run the syslogConf.pl script, ensure that the Daemon Manager is stopped.

Running the syslogConf.pl Script

To run the script:

Step 1 Go to the command prompt and enter:

```
NMSROOT/bin/perl NMSROOT/syslogConf.pl
```

When you run this syslogConf.pl script, a message appears with five options.

```
[1] Change Syslog Analyzer Port
[2] Change Syslog Collector Port
[3] Configure Remote Syslog Collector(RSAC) Address and Port
[4] Change Syslog File Location
[Q] Quit
Enter Your Choice:
```

Step 2 Enter your choice.

- If you enter **1** the following message is displayed with the old Syslog Analyser Port number. You are also prompted to enter the new port number for the Syslog Analyser.

  ```
  INFO: You have opted to change Local Syslog Analyser port.
  Old Syslog Analyser Port :xxxx
  Enter new Syslog Analyser port:
  ```

  For example, you can change the Syslog Analyser Port from 4444 to 2222.

  After providing the new port information, the following message is displayed.

  ```
  INFO:Local Syslog Analyser port has been changed from 4444 to 2222 successfully
  ```

- If you enter **2** the following message is displayed with the old Syslog Collector Port number. You are also prompted to enter the new port number for the Syslog Collector.

  ```
  INFO: You have opted to change Local Syslog Collector port.
  Old Syslog Collector Port :xxxx
  Enter new Syslog Collector port:
  ```

  For example you can change the Syslog Collector Port from 1111 to 3333.

  After providing the new port information, the following message is displayed.

  ```
  INFO:Local Syslog Collector port has been changed from 1111 to 3333 successfully
  ```

- If you enter **3**, the following message is displayed, with the old Syslog Collector Port number. You are also prompted to provide the new RSAC Address and the new port number for the Syslog Collector.

  ```
  INFO: You have opted to change RSAC port.
  Enter the RSAC Address:
  Old Syslog Collector Port :0
  Enter new Syslog Collector port:
  ```

  Ensure that the RSAC port that you configure in the LMS server corresponds with the Collector port configured in the RSAC server.

  You can specify srme-w2k as the RSAC Address, and change the Syslog Collector port from 0 to 3456.

  After providing the RSAC Address and port information, the following message is displayed.

  ```
  INFO: Remote Syslog Collector(RSAC) port has been changed from 0 to 3456.
  ```

- If you enter **4**, the following message is displayed with the old Syslog Directory Path. You are also prompted to enter the new Syslog Directory path.

  ```
  INFO: You have opted to change Syslog File Location
  Old Syslog Directory : /var/log/
  Enter Full Path of New Syslog Directory:
  ```

  Ensure that you enter the full directory path, if you are running the syslogConf.pl script on Solaris and Soft Appliance. You can provide the relative directory path if you are running the syslogConf.pl script on Windows.

  For example you can change the Syslog Directory location from /var/log/ to /var/log/newSyslogLoc.

  After providing the required information, the following message is displayed.

  ```
  Syslog file location changed from: /var/log/ to: /var/log/newSyslogLoc
  ```

- If you enter **Q,** you are allowed to quit from the script.

Software Management CLI Utility

You can invoke Software Management (SWIM) features from Command Line Interface (CLI).

The **`cwcli swim`** commands let you use SWIM features from the command line. You can use the **`cwcli swim`** commands to:

- List Images from Software Management (SWIM) Repository
- Export Images from Software Management (SWIM) Repository

These functions are only accessible to the Network Administrator, Network Operator and super users who have all of the roles.

If you do not have permission to run custom commands, you can run a command or command set from the CLI only if:

- The command set is assigned to you by the Administrator.
- The command set has at least one command that can be run on the specified device.

This section contains:

Running cwcli swim Command

The command syntax for running **`cwcli swim`** commands is:

`cwcli swim` *`subcommands common_arguments command_arguments`*

In the CLI version, you can provide the arguments in the (operating system shell) command line or in aninput file.

The input file gives you flexibility and control over commands and command sets. You can specify the images on which you want to run the command sets.

In the input file, you can include subcommands and command arguments.

Items in square brackets ([]) are optional; items in curly brackets ({}) are required.

The arguments are described in the following sections.

Subcommands

Subcommands specify the actions that you perform. Valid values for subcommands are described in the following table.

Sub Command	Description	Example
`listimages`	Displays a list of images available in the Software Repository.	`cwcli swim listimages -u` *Userid* `-p` *Password*
`exportimages`	Exports specified images in a non-compressed format from the Software Repository to any directory. The default target directory is the current directory. For exportimages command either one of these arguments is mandatory: `-imagenames` | `-all` | `-input`	`cwcli swim exportimages -u` *Username* `-p` *Password* [`-imagenames` *imagename1*, *imagename2*...] [`-all`] [`-dirname` *directoryname*] [`-input` *argumentFile*] [`-m` *email*][`-l` *logfile*]
`help`	Displays command usage information.	`cwcli swim -help`

Common Arguments

Common arguments specify parameters that apply to all subcommands. Valid values for*common_arguments* are described in the following table.

Command Arguments	Description/Action	Usage Notes
`-u` *user*	Enter a valid LMS username.	None
`-p` *password*	Enter the password for the username. You can also specify the password in a file. See Setting CWCLIFILE Environment Variable for more details.	None
[`-l` *log_filename*]	Identifies a file to which Software Management Commands will write log messages. If you do not specify this, the log output will appear on screen.	This argument is optional. *log_filename* can be a full path to the file or a filename in the local directory. If you do not specify filename, the log file will be created in: • /var/adm/CSCOpx/log (On Solaris and Soft Appliance) • *NMSROOT*\log (On Windows) *NMSROOT* is the LMS install directory.
[`-m` *Email ID*]	Enter your Email ID	This argument is optional. You will get the output of the CLI operation in an e-mail.

Command Arguments

Command arguments specify parameters that apply only to specific subcommands. Valid values for command arguments are described in the following table.

Arguments in square brackets ([]) are optional. Arguments in curly brackets ({}) are required. You must provide one argument from each group of arguments in curly brackets ({}) that is separated by vertical bars (|).

Command Arguments	Description	Usage Notes
Command Arguments for `exportimages`		
{**`-imagenames`** *ImageName1, ImageName2*}	Specify the image names that you want to export using this command.	**`cwcli swim exportimages -u`** *Username* **`-p`** *Password* [**`-imagenames`** *imagename1*, *imagename2*...] [**`-all`**] [**`-dirname`** *directoryname*] [**`-input`** *argumentFile*] [**`-m`** *email*][**`-l`** *logfile*] *ImageName1, ImageName2* —List of images. Separate these names by commas.
{**`-all`**}	Specify this option if you want to export all images from Software Repository to the current directory or any specified directory.	--
{**`-input`** *argumentFile*}	Input file containing the details of the subcommands	If you are specifying the input file, you need not specify the subcommands. For instance, if you are using sample.txt as the argumentFile for -input command, you have to provide the following command: **`cwcli swim exportimages -input`** *sample.txt* Example of sample.txt: **`-imagenames`** {*imagename1}*, [*imagename2*...] **`-imagenames`** {*imagename4}*, [*imagename5*...] Items in square brackets ([]) are optional; items in curly brackets ({}) are required.
[**`-dirname`** *directoryname*]	Specify a directory name if you want to export images to a specified directory using this command.	If you do not specify this the images are exported to the NMSROOT/temp directory, where CLI is launched.

Running SWIM CLI Remotely

You can run Software Management (SWIM) CLI from a remote console.

SWIM uses the Remote Access feature in the CLI framework to help you to invoke the SWIM commands from the client in the same way as you run them on the LMS server.

The name of the servlet to be invoked is /rme/cwcli.

You must invoke the following URLs to run any command.

- For POST request:

  http://*lms-server:lms-port*/rme/**cwcli** *payload XML file*

- For GET request:

  http://*lms-server:lms-port*/rme/cwcli?command=**cwcli swim** *command* **-u** *Username* **-p** *Password command_specific_args*

The contents of the payload.xml is:

```
<payload>
<command>
cwcli swim command -u Username -p Password command_specific_args
</command>
</payload>
```

For example to execute the **listimages** command payload.xml will be as follows:

```
<payload>
<command>
cwcli swim listimages -u Username -p Password
</command>
</payload>
```

Note The *Base64* encoded password is used for accessing Software Management (SWIM) CLI remotely.

To invoke the servlet using a script, see the Overview: cwcli invreport Command.

The script and the payload file should be residing in the client machine.