1 Introduction................................................................. 5
  1.1 Directives and Components........................................... 9
    1.1.1 Clusters with 64 or fewer data servers.......................... 10
    1.1.2 Clusters with 65 or more data servers.......................... 12
    1.1.3 Frequently asked questions......................................... 15
  1.2 Before you get started.................................................. 17
  1.3 Starting the cmsd Process.............................................. 19
    1.3.1 Multiple Instances and Automatic Fencing........................ 24
    1.3.2 Log File Plug-Ins.................................................. 25
    1.3.3 Files created by cmsd.............................................. 26
    1.3.4 Exported Environment Variables................................... 27
2 Mandatory Configuration Directives................................. 29
  2.1 manager ........................................................................ 29
    2.1.1 Choosing all vs. any for Normal Managers......................... 32
    2.1.2 Peer Manager File Location and Selection.......................... 33
    2.1.3 Disjoint Cluster Configurations...................................... 35
  2.2 role .............................................................................. 37
    2.2.1 Role Summary Table.................................................... 38
3 Common Configuration Directives .................................... 41
  3.1 allow.............................................................................. 41
  3.2 defaults (an oss directive)............................................... 43
  3.3 dfs.................................................................................. 45
  3.4 export............................................................................. 51
  3.5 localroot (an oss directive)............................................. 53
  3.6 perf................................................................................. 54
  3.7 prep................................................................................. 57
    3.7.1 Optional Prepare Interface Program Requirements.............. 59
  3.8 sched .............................................................................. 63
  3.9 seclib.............................................................................. 69
  3.10 space............................................................................. 70
  3.11 space (an oss directive).................................................. 72
4 Esoteric Configuration Directives...................................... 73
  4.1 altds ............................................................................. 73
  4.2 blacklist .......................................................................... 74
  4.3 cidtag ............................................................................. 76
  4.4 conwait ........................................................................... 77
  4.5 delay ............................................................................... 79
    4.5.1 Relationship Between hold & lookup Delay vs. qdl.............. 83
Contents

4.6  fxhold.................................................................................................84
4.7  fsseq..................................................................................................85
4.8  namelib (an oss directive).................................................................87
4.9  nbsendq.............................................................................................89
4.10 nowait.................................................................................................91
4.11 osslib (an ofs directive).................................................................92
4.12 pidpath...............................................................................................93
4.13 ping..................................................................................................95
4.14 prepmsg.........................................................................................97
4.15 remoteroot (an oss directive).........................................................99
4.16 repstats..........................................................................................100
4.17 request.............................................................................................101
4.18 subcluster.......................................................................................102
4.19 superport ......................................................................................103
4.20 vnid..............................................................................................104
  4.20.1 Using Virtual Network Identifiers .............................................105
  4.20.2 Virtual Network Identifiers and Kubernetes .........................107
4.21 trace...............................................................................................108
4.22 whitelist..........................................................................................109

5  Blacklist and Whitelist File Format .................................................111

6  Document Change History ..............................................................113
1 Introduction

This document describes Cluster Management Service Distributed configuration directives. This component provides dynamic load balancing of files and namespace consolidation of distributed data regardless of location. The cms (Cluster Management Services) component is meant to be used with xrootd’s Open File System (ofs) component. Refer to the “XRootD ofs & oss Reference” for detailed information.

Directives for cmsd, the clustering daemon and its client counterpart used by the ofs component, come from a configuration file. The configuration file is mandatory and its location is specified on the command line using the -c option.

The characters cms must prefix each cmsd-specific directive in the configuration file. Directives that apply to multiple components must be preceded by the characters all. This makes cmsd directives compatible with the xrootd’s other configurable components and allows you to use a single configuration file. Having a single configuration file is important because the cmsd must also inspect directives destined to the oss (Open Storage System) component. These are prefixed by oss. Having one configuration file makes this task transparent.

Records that do not start with a recognized identifier are ignored. This includes blank record and comment lines (i.e., lines starting with a pound sign, #). This guide documents the all, cms and oss configuration directives that are relevant to the cmsd.

Refer to the manual “Configuration File Syntax” on how to specify and use conditional directives and set variables. These features are indispensable for complex configuration files usually encountered in large installations.

Clustering is performed by a set of cooperating servers. One or more cmsd daemons run in manager mode and can be used by one or more xrootd’s to determine where to redirect a client’s file request. The request can only be redirected to a machine that is running a cmsd in server or supervisor mode. There can be up to 64 cmsd servers. Each machine can run one or more xrootd’s. The following figure illustrates a simple minimal system.
In the diagram, there are three hosts: \( x \), \( y \), and \( z \). Host \( y \) serves as the redirector. Hosts \( x \) and \( z \) are the hosts that can be used to serve data to clients. Consequently, host \( y \) runs a manager \texttt{cmsd} while hosts \( x \) and \( z \) run server \texttt{cmsd}’s.

The servers connect to the manager and provide load and file information. The \texttt{xrootd} running on host \( y \) connects to the manager as well. However, the \texttt{xrootd} on host \( y \) uses the manager to determine which server to direct client requests. It does not serve any actual data files.

The typical open request is handled in four steps:

1. The client directs the open request to the \texttt{xrootd} that runs on the manager’s host.
2. The \texttt{xrootd} asks the \texttt{cmsd} manager which machine is the best to use to process the file. The manager determines the best machine using a variety of configurable parameters.
3. The \texttt{xrootd} on host \( y \) tells the client, in this example, that host \( z \) is the best host to use for the file.
4. The client then redirects the request to the \texttt{xrootd} running on host \( z \).
In order to make the system as flexible as possible, the manager `cmsd` does not know how many or which hosts will act as servers. For security purposes, you can restrict hosts based on host name as well as by `NIS` netgroup. Thus, servers essentially subscribe to the manager claiming that they have file resources. During the subscription process, each server indicates the file paths to which it is willing to provide data access. Periodically, the manager `cmsd` requests load information from each server. Each server reports CPU, network I/O, queue, memory, paging load as well as free space. This information is used to select the best available server for an open request.

The decision is tempered whether or not the server already has the file on disk or whether the file must be staged to disk from a Mass Storage System. The manager may decide that all available servers are too loaded and force a file to be replicated on a less loaded server. This provides additional data paths to the file. Replicated load balancing is only compatible with read-only files. The manager can direct client’s to a writable version of a file but only on servers that have indicated that they offer write access on the associated path. In general, only one such server may exist for each particular path.

In order to provide a fully redundant service, all servers may be replicated and cross-connected, as full full crossbar configuration shows above.
Each server `cmsd` subscribes to two manager `cmsd`'s. Each `xrootd` that can redirect clients subscribes to two managers. Thus, the loss of any single manager `xrootd` does not affect load balancing. More complex arrangements are possible since each server may have any number of managers and each `xrootd` can subscribe to any number of managers.

In order to ease migration, any peripheral (i.e., data server) `xrootd` can always be directly used. This means that redirection only occurs when a client contacts a redirecting `xrootd`. For systems that are being configured this way for the very first time, you should always use the “`xrd.port any`” directive for data server `xrootd`'s. This allows the `xrootd` to use an arbitrary port number. In this mode it is very difficult for any client to directly use a data server `xrootd` without first contacting the manager `xrootd` first.
1.1 Directives and Components

Clustering consists of four distinct components:
1. The manager cmsd process (typically in a separate machine).
2. Supervisor cmsd processes (only for clusters of more than 64 servers).
3. Server cmsd processes, and
4. An integrated cmsd client in the xrootd process, which can be a manager, supervisor, or server.

A manager cmsd always communicates with supervisor and server cmsd’s as well as a manager xrootd. Server xrootd’s only communicate with their server cmsd counterpart. Two distinct directives are used to identify the participants:
1. **all.role** which tells each component whether it is to function as a manager, supervisor, or server; and
2. **all.manager** that tells each component the DNS name of the manager.
1.1.1 Clusters with 64 or fewer data servers

Use the following general steps to successfully configure a cluster that has 64 or fewer data servers:

- Determine which servers will be used for data serving (i.e., run a data server \texttt{cmsd}) and which for redirection (i.e., run a manager \texttt{cmsd}). A manager is not capable of also serving data. Use the \texttt{all.role} directive to differentiate servers and managers.
- Use the \texttt{allow} directive to restrict the set of valid data servers.
- Determine the scheduling policy using the \texttt{cms.sched} and \texttt{cms.space} directives.
- Determine which port number will be used for inter-server communication and specify it for the manager \texttt{cmsd} using the \texttt{mandatory all.manager} directive (see below).
- Remember to specify the \texttt{xrootd} port number for the associated manager \texttt{xrootd} using the \texttt{xrd.port} directive, if the default of 1094 is unacceptable.
- Determine the set of data servers. Unless direct access is important, you should configure data servers with \texttt{xrd.port any}.
- For each data server, determine which file paths it will handle. In general, a number of servers should serve the same path.
- Use the \texttt{all.export} directive on each data server to restrict it to its set of paths.
- If a data server will be also creating files, use the \texttt{space} directive to indicate which file systems may be used for file creation.
- Use the \texttt{all.manager} directive to tell each \texttt{xrootd} and \texttt{cmsd} the location of its set of managers.
- Start a data server \texttt{cmsd} server on each \texttt{xrootd} data server machine. For each \texttt{xrootd} that will be redirecting, use the \texttt{all.role manager} directive to enable redirection mode.\footnote{Refer to the “xrootd ofs & oss Configuration Guide” for more information.}
- Start the manager, server, and \texttt{xrootd}. See the \texttt{cmsd} command described in the next section. The start-up order is not important.
The following diagram and corresponding configuration file illustrates how to cluster 30 data servers with two managers.

```bash
# Specify the data server port number. This is only relevant to
# managers, so we qualify the specific port number using the "if".
#
xrd.port any
xrd.port 1094 if man01.u.org man02.u.org

# Specify which paths are to be exported (default is r/w)
#
all.export /data

# Tell everyone the role it will have. Use a default of server but
# qualify it depending on hostname using the "if".
#
all.role server
all.role manager if man01.u.org man02.u.org

# Tell everyone the location of each manager.
#
all.manager man01.u.org:1213
all.manager man02.u.org:1213

# Tell the cmsd which machines are allowed to connect
#
cms.allow host man*.u.org
cms.allow host data*.u.org
```

**Configuration “myconfig.cf” for a 30 Data Server Cluster**

There are additional directives to further tune the system and are described on the following pages.
1.1.2 Clusters with 65 or more data servers

Configuring a cluster of more than 64 data servers is just slightly more complicated than configuring a smaller cluster. The complication arises from the fact that some additional management servers need to be started. The configuration file, however, is no more complicated. Below are the steps you should take to successfully configure large clusters.

- Choose the port numbers you wish to use for the manager \texttt{xrootd} and \texttt{cmsd} servers. Typically, \texttt{xrootd} uses port 1094 and specified with the \texttt{xrd.port} directive. For \texttt{cmsd} port 1213 is used and specified with the \texttt{all.manager} directive. Other \texttt{xrootd} servers should specify “\texttt{xrd.port any}”.

- Choose the number of manager nodes you wish to run. You must configure at least one manager node. The manager is the first point of contact for a client and is also the cluster leader. A manager should run on a dedicated machine of modest power (e.g., 2GB RAM, 1GHz clock speed, 100Mb ethernet).

A manager node consists of
- an \texttt{xrootd} configured with the “\texttt{all.role manager}” directive.
- a \texttt{cmsd} configured with “\texttt{all.role manager}” directive.

You may configure more than one manager and run them in either fail-over mode (the default) or in load balancing mode where each manager shares part of the client load (see the \texttt{all.manager} directive). Each manager \texttt{xrootd-cmsd} pair must run on a separate machine.

- Compute the number of supervisor nodes you need. A supervisor node acts as a local manager for a group of 64 other nodes. These nodes may be data servers or supervisors. A supervisor node consists of
  - an \texttt{xrootd} configured with the “\texttt{all.role supervisor}” directive.
    Additionally, specify the “\texttt{xrd.port any}” directive.
  - a \texttt{cmsd} configured with the “\texttt{all.role supervisor}” directive.

You only need to configure supervisor nodes if you are running more than 64 data servers. The number of supervisor nodes is based on the number of available manager plus supervisor slots. A recursive formula is needed to calculate the minimum number. Since you normally wish to start more than the minimum number of supervisors, a simplified formula can be used.
Conservatively, you will need one supervisor node for each group of 64 data servers. For instance, if you plan to run 500 data servers you will need the upper limit of 500/64 supervisors (i.e., 8).

Each supervisor node can run on a data server node. If you wish to share resources in this way, choose data server nodes that will be as lightly loaded as possible. The performance requirements for a supervisor node are the same as a manager node.

- Configure the data server nodes. A data server node delivers actual data to clients. It consists of
  a) an **xrootd** configured with the **"all.role server"** directive. Additionally, specify the **"xrd.port any"** directive.
  b) a **cmsd** configured with the **"all.role server"** directive.

Configure as many data server nodes as you need. Keep in mind that at least one additional supervisor node is need for each group of 64 data servers.

The performance requirements are determined by the performance needs of clients. The server should have enough disk space, adequate network bandwidth (e.g., Gb ethernet), and significant cpu and i/o resources. If you wish to use memory mapped files, then the node should have a commensurate amount of real memory.

For example, assume you wish to cluster 99 data servers in the way shown below.

For example, assume you wish to cluster 99 data servers in the way shown below.

Here we wish to have only one manager. We will need at least one supervisor. While the simplistic formula indicates two supervisors are needed; in practice, the cluster could self-organize by affiliating 63 data servers and one supervisor (a total of 64) with the manager and affiliating the remaining data servers (36) with the supervisor.
With two supervisors, the cluster would affiliate 62 data servers and two supervisors with the manager, and split the remaining data servers across the two supervisors. So, either configuration would work. Fortunately, the cluster attempts to automatically find the best organization given the resources at hand. Configuration files for small and large clusters will differ only slightly from each other. Notable differences involve `allow` and `role` directives. Configuration file simplicity relies on the use of regular names for various hosts.

```
# Specify the data server port number. This is only relevant to managers, so we qualify the actual port number using the "if".
#
# xrd.port any
xrd.port 1094 if man01.u.org

# Specify which paths are to be exported (default is r/w)
#
all.export /data

# Tell the cmsd which machines are allowed to connect
#
cms.allow host man01.u.org
cms.allow host sup01.u.org
cms.allow host data*.u.org

# Indicate the role this server will have based on host name (the default role is that of server)
#
all.role server
all.role supervisor if sup01.u.org
all.role manager if man01.u.org

# Tell everyone the location of the manager.
#
all.manager man01.u.org:1213
```

Configuration “myconfig.cf” for a 99 Data Server Cluster
1.1.3 Frequently asked questions

Does start-up order matter?
Generally, it does not matter in which order nodes are started. For the efficiency minded, starting supervisor nodes ahead of data server nodes allows the system to converge on a stable configuration faster.

How long will it take for the system to converge?
This depends on how many servers are in the configuration. Generally, it takes approximately 1 to 13 seconds for a server to find its correct place in the cluster. However, the process is run in parallel across all of the servers. So, the system should converge in less than 30 seconds for a configuration of about a 1,000 nodes. By default, the system delays full availability for 90 seconds, this should be sufficient time for convergence of even extremely large installations.

What happens if I have too few supervisors?
If there are not enough supervisors relative to the number of data servers, one or more data servers will be orphaned and unavailable. If you suspect this, check the manager’s log. It will contain warnings about orphaned data servers.

What happens if I have more supervisor nodes than I need?
Since the system tries to evenly distribute data servers across all available supervisors, excess supervisors are used to further reduce the load on supervisor nodes. The excess supervisors are also used as “hot spares” in the event one of the supervisors becomes unavailable. You should configure as many “extra” supervisors as you feel are necessary to provide a suitable level of fault tolerance.

Can I run all the supervisors on a single node?
Yes, but you will need to assign each cmsd a unique instance name using the –n option. Additionally, the same –n option value must be specified for the xrootd that is paired with a particular cmsd. Use the “if” directive, keyed off the instance name, to maintain a single configuration file. Finally, each xrootd, other than the one tied to the manager cmsd, must be started with the “port any” directive to allow for arbitrary port selection. You should realize that running all of the supervisors on a single node creates a large single point of failure.
How do I run a data server and a supervisor on the same node?

Use the provided StartCMS and StartXRD scripts. For a supervisor cmsd and xrootd, specify the “all.role supervisor”. For a data server cmsd and xrootd specify the “all.role server” directive. You should make sure that “xrd.port any” is specified for supervisor and data server xrootd’s to prevent any port conflicts.

What does the “–port any” xrootd command line option actually do?

The “–port any” option allows xrootd to choose any port that is available. The selected port number is then forwarded to the cmsd. This allows the cmsd to redirect clients to the proper port even though it’s not known ahead of time. This only works if the cmsd is not started with the -i option (the default) and the xrootd is started with the “all.role server” (for data servers) or all.role supervisor directive (for supervisors). This does not eliminate the need for starting the manager cmsd and its xrootd counterpart with well-known ports.

Does that mean I can use –port any to run multiple data servers on a single node?

Yes. See the answer to “Can I run all the supervisors on a single node?”

Can I use the –port any option to prohibit clients to bypass the cmsd?

Yes. This is actually recommended. Since arbitrary port numbers are chosen, a client cannot directly connect to a data server without using the manager xrootd. However, while significant programming effort is required to capture port numbers at run-time; any “management by obscurity” method can be defeated.

I want to run the cluster using Kubernetes, are there any admonitions?

Yes. Many container orchestration systems have assumptions that run counter to running on bare hardware or even a standalone container. You should do the following to avoid odd behavior:

a) Specify the dyndns option via the xrd.network directive.

b) Specify the cms.vnid directive for all data servers (xrootd and cmsd).
Is there a preferred way of stopping a cmsd or xrootd process?

No. The simplest and most effective way to stop any XRootD component is to use the kill command. The system is architected to withstand catastrophic failure (e.g. a power failure). Please remember, when you kill a cmsd on a data server node. Its redirector (i.e. manager or supervisor) will remember it for a configurable amount of time (default is 10 minutes). If that server had the only copy of a requested file, the redirector will delay the client with the hope the server comes back. After the wait time is over and the server still has not come back, the client will get a “file not found” error.

1.2 Before you get started…

It’s best to configure the xrootd daemon first and make sure it works as a regular server. Configuring an xrootd daemon is much simpler than configuring a cmsd. Plus, it gives you the opportunity to become familiar with how directives work; especially the xrd directives.

The cmsd uses the xrd framework to drive all of its activities; so the xrd directives directly control how the cmsd interacts with its environment. Whatever you choose for the xrootd is likely to be best for the cmsd. Thus, you only need to go through the exercise once.

Many of the command line options used for the xrootd also apply to the cmsd, thus once you have determined the best settings for the xrootd you can carry over many of those choices to the cmsd as well.

Finally, use a single configuration file for the xrootd and cmsd daemons. It makes life far simpler and avoids making inconsistent choices between two configuration files. In cases where a different choice is needed; use the if-else-fi directives to special case your choices.
1.3 Starting the cmsd Process

Use the following command to start a manager or server cmsd process.

```
cmsd -c cfn [-l largs] [-k {num | sz{k|m|g} | sig}] [opts]
```

**Parameters**
- `-c cfn` The name of the configuration file. You must specify the name of a configuration file even if it is empty.

**Options**
- `-l largs` Specifies how messages are to be handled. Options are:
  - `fn` Directs messages and any trace output to the indicated file, `fn`, possibly qualified by the instance name (see the fencing section). By default, messages are directed to standard error.
  - `=fn` Same as `fn` but the `fn` is not qualified by the instance name, if any. This allows log files to be handled in an arbitrary manual way. For more information see the section on fencing.
  - `@lib` Directs messages to a plug-in that is defined in the shared library specified by `lib` (see the section on log file plug-ins). Additional comma-separated parameters may follow `lib`, as follows:
    - `bsz=sz` Specifies the size of the speed matching buffer. The default is 64K. Messages are placed in the buffer and then forwarded to the plug-in as time permits. A value of 0 disables speed matching and messages are handed off to the plug-in as they occur. See the section on log file plug-ins for more information. A positive value less than 8K is forced to be 8K. The maximum allowed in one
megabyte. The sz may be suffixed by k or m to indicate kilobytes or megabyte, respectively.

cse={0 | 1 | 2} Specifies how standard error output should be handled:
0  Does not capture standard error output. All such output is sent to the logfn destination, if specified, or is otherwise lost. This is the default.
1  Captures standard error but only forwards it to the logging plug-in if it starts with a standard time stamp. This option may cause an infinite loop. Refer to the logging plug-in section for more information.
2  Captures standard error output and forwards it to the logging plug-in without inspection. Refer to the logging plug-in section for more information.

logfn=[|=]fn Specifies that messages are also to be routed to a local log file. The parameter is identical to that described above. To use standard error, specify a dash (-) for fn.

-k num | sz[k|m|g] | sig
Keep no more than num old log files. If sz is specified, the number of log files kept (excluding the current log file) is trimmed to not exceed sz bytes. The sz must be suffixed by k, m, or g to indicate kilobytes, megabyte, or gigabytes, respectively. If a sig value is specified (i.e. hup etc), then an external program is expected to handle log file rotation (e.g. logrotate). Except for fifo, the argument specifies signal that causes the daemon to close and re-open the log file to allow rotation to occur. When fifo is specified, the daemon waits for data to appear on a fifo whose path is identical to the log file path but whose name is prefixed by a dot. Refer to the notes for manual rotation caveats.

Esoteric Options
{-a | -A} apath
Specifies the default administrative path and can be overridden by the adminpath directive in the configuration file. When -A is specified group write access is allowed (see the adminpath directive group option in the Xrd/Xrootd reference for details).

-b  Runs the program in the background. You should also specify -l.

-d  Turns on debugging. Warning! This severely impacts performance.
The **cmsd** subscribes to a manager **cmsd** whether or not the local primary data server contacts the **cmsd**. Also, see the **cms.nowait** directive.

**-I **{v4 | v6}**
Restricts the server’s internet address protocol. When v4 is specified, only hosts with **IPV4** addresses can connect or be connected to. When v6 is specified, the default, hosts using **IPV6** or **IPV4** addresses can connect or be connected to. This option is only useful for systems that have misbehaving **IPV6** network stacks. The default is established by the network interface configuration on the machine at the time the program starts.

**-n name**
The instance name of the **cmsd**. There is no default. See the notes for more information on this option.

**-p port**
The TCP port, or service name associated with a port, that the manager **cmsd** is to use for new connections. There is no default. If the port is not specified on the command line, it must be specified using the **all.manger** directive.

**-s pfn** Specifies the name of the file that is to hold the process id upon start-up.

**-S site** Specifies a 1- to 15-character site name that is to be included in monitoring records. The name may only contain letters, digits and the symbols “-_:.”; any other characters are converted to a period.

**{-w | -W} hpath**
Specifies the default home path; i.e. the current working directory during execution. If it is not specified on the command line, it can be specified by the **homepath** directive in the configuration file. When -W is specified group read access is allowed (see the **xrd.homepath** option for details). The hpath is extended by any specified instance name (i.e. -n option). The path is created should it not exist.

**-z** provides microsecond resolution for log file message timestamps.

**Defaults**
**cmstd** -l -
Notes

1) A configuration file is not optional.
2) The same configuration file may be used for manager and server cmsd’s. Directives not relevant to a particular mode of operation are ignored.
3) The cmsd related directives may be placed in the xrootd configuration file as well. Thus only one configuration file needs to be maintained per machine.
4) The order in which servers are started is unimportant.
5) When a signal value is specified, log files are not automatically renamed at midnight. Instead an external program must be used to properly rotate log files. Make sure to choose a signal that is not in use by any plug-in. If unsure, choose one of the obscure signal names and monitor for any odd behavior. Otherwise, use the fifo option. Be aware that on some non-Linux platforms the fifo file descriptor may leak.
6) When fifo is specified the fifo file name must not exists or exist as a fifo file. A simple “echo x >> /path/lfn” causes the logfile to close and reopen.
7) The sig names, except for fifo, be fully capitalized as well prefixed by “sig” or “SIG” when capitalized.
8) You must start at least one cmsd in manager mode. The number of supervisor cmsd’s is approximately determined by dividing the number of server mode cmsd’s by 64 less one.
9) In a supervisor role, the cmsd acts as both manager and server. Supervisor cmsd’s are used to cluster groups of 64 server cmsd. Since a supervisor cmsd can subscribe to a manager or supervisor cmsd, it is possible to cluster together a virtually unlimited number of data servers.

Notes on Esoteric Options

1) Whenever the companion xrootd looses contact withits cmsd, the host automatically becomes ineligible for selection until it reconnects
2) The –i option provides for a loose coupling between servers running on the same host. The cmsd executes asynchronously from the host’s data server and can subscribe to a manager before the data server is available on the host.
3) The –i option is meant for to be used with data servers that are unable to communicate with the local cmsd. You should not specify this option for the xrootd server.
4) Warning: the default cmsd mode (i.e., wait for data server) must be used in conjunction with xrootd’s configured for clustering; otherwise the host will never be selected by the manager cmsd.
5) **Warning:** The –i option disables port remapping. With port remapping, a client is redirected to the port actually being used by the data server that is the target of the redirection. This allows arbitrary or hidden ports to be used, none of which need be the same. When port remapping is disabled, clients are always redirected to the port they initially used to contact the redirector.

6) The –b option forces the program into the background. If –l is not specified; all output messages are discarded.

7) The -a, -b, -p, and -s command line options are meant to be used by start-up scripts (e.g. init.d or systemd).

8) **Warning:** Command line options, except for –a and -s, over-ride corresponding configuration file directives.

**Example**
```
cmsd -c /opt/xrootd/cmsd.cf
```
1.3.1 Multiple Instances and Automatic Fencing

The `cmsd` supports running as many `cmsd`’s as you would like on the same host (i.e., machine). This is accomplished by the `-n` command line option. This option assigns an instance name to the `cmsd`. The `cmsd` uses instance name to maintain a separate disk name space for files that it needs to create.

There is no default instance name; however, the system uses the word `anon` to refer to unnamed `cmsd`’s. By design, there can only be one logical instance combination of a manager, supervisor, and server running on the same machine. The `-n` option allows you to create new logical instances by assigning each instance a different name. This allows you to run multiple instances of the `cmsd` on the same machine.

Server and supervisor `cmsd`’s pose no port contention problems since they always use whatever port happens to be free. Manager `cmsd`’s are assigned specific port numbers (see the `manager` directive). Therefore, if you wish to run more than one `cmsd` manager on a host, each manager must also be assigned a unique port number.

The `cmsd`’s always work in pairs with `xrootd`’s. The pairing only works within the same instance. That is, if a `cmsd` with an instance name of `foo` is to be used with a particular `xrootd`; then that `xrootd` must be given an instance name of `foo` as well. Additionally, the `cmsd` and `xrootd` home directories should differ to avoid core file conflicts.

Failure to follow these directions will prevent proper communications from being established between `xrootd`’s and `cmsd`’s.

Once an instance name is assigned to a daemon using the `-n` option, the system automatically fences in the daemon so that it does not interfere with any other `xrootd` processes running with it. Automatic fencing consists of these actions:

- The instance name is suffixed to the `adminpath` to create a unique location for temporary server files. For instance, if `-n` is not specified, `xrootd` creates `/tmp/.xrootd/admin` as the path for the administrative interface. If “-n test” is specified, `xrootd` creates `/tmp/test/.xrootd/admin` instead. Even the path specified with the `adminpath` configuration directive is modified.
• The instance name is used to create a new directory in the current working directory. The current working directory is changed to this newly created path. So, if “/home/xrootd” is the current working directory and “-n test” is specified; the current working directory becomes “/home/xrootd/test”. This allows core files to be segregated by instance name.

• The instance name is automatically inserted into the log file path specified via the –l command line directive to create a unique location for server log files files. For instance, if “–l /var/adm/xrootd/cmslog” is specified along with “-n test”, cmstd modifies the –l argument to be /var/adm/xrootd/test/cmslog.

Automatic fencing of log files may, for some installations, run counter to the way log files are commonly handled. You can disable fencing of log files by prefix the log file path by an equals sign. However, you are then responsible to make sure that each instance uses a different log file path or name.

1.3.2 Log File Plug-Ins
XRootD allows you to specify a plug-in to handle messages that would otherwise be sent to a regular file or standard error. You do this using the ‘@’ qualifier with the –l option. Logging messages is a critical function in the server and any delay will severely impact server performance. The default logging path is very efficient and any plug-in placed in the path should be just as efficient. To help, a speed matching buffer is used to minimize plug-in vagaries. However, if you choose to not use a speed matching buffer (i.e. a bsz of zero for synchronous operation) then the plug-in becomes the choke point in server performance.

You may also choose to capture standard error output using the cse parameter. However, this option will result in an infinite loop if your logging plug-in writes to standard error for any reason. This may be mitigated by specifying cse=1 which only sends standard error output to the plug-in if it starts with a timestamp of the form “ymmd hh:mm:ss”. All debugging output starts with such a timestamp.

The details on how you write a plug-in is detailed in the XrdSysLogPI.hh header file. It is important to realize that if you use the XrdSysLogger object to route a message from your plug-in, an infinite loop will result. Additionally, one log file plug-in is used to all XrdSysLogger instances.
1.3.3 Files created by cmsd

The following files are created by the cmsd:

<table>
<thead>
<tr>
<th>Default File</th>
<th>Type</th>
<th>Modified by</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;stderr&gt;</code></td>
<td></td>
<td>-l option and -n option</td>
<td>Informational and error messages</td>
</tr>
<tr>
<td>/tmp/[name]/olb/olbd</td>
<td>TCP Socket</td>
<td>adminpath</td>
<td>Local xrootd - server cmsd communications</td>
</tr>
<tr>
<td>/tmp/[name]/olb/olbd.super</td>
<td>TCP Socket</td>
<td>adminpath</td>
<td>Local xrootd - supervisor cmsd communications</td>
</tr>
<tr>
<td>/tmp/[name]/olb/olbd.notes</td>
<td>UDP Socket</td>
<td>adminpath</td>
<td>Local cmsd server event notifications</td>
</tr>
<tr>
<td>/tmp/[name]/olbd/olbd.seton</td>
<td>UDP Socket</td>
<td>adminpath</td>
<td>Local cmsd supervisor event notifications</td>
</tr>
<tr>
<td>/tmp/[name]/cmsd.pid</td>
<td>File</td>
<td>pidpath and -n option</td>
<td>Holds the process id.</td>
</tr>
<tr>
<td>&lt;cwd&gt;/%[name]/core[.pid]</td>
<td>File</td>
<td>-n option</td>
<td>Core file.</td>
</tr>
<tr>
<td>/tmp/xrootd.[name].env</td>
<td>File</td>
<td>Adminpath</td>
<td>Holds environmental information (see the xrd/xrootd reference).</td>
</tr>
</tbody>
</table>

1.3.3.1 Environmental Information File

The daemon writes environmental information in the directory specified by the –s command line directive and if not specified, in /tmp. This information can be used to automatically collect all relevant information about a daemon to facilitate automatic problem resolution.

The environmental file is named “cmsd.[name].env” where name is the instance name and anon if no instance name was specified. The format of the information is shown below. When parsing this information, you should not depend on the order shown below.
1.3.4 Exported Environment Variables

The following table shows the environment variable exported by *xrootd*. These may be used by external programs and plug-ins, as needed. They should never be modified.

<table>
<thead>
<tr>
<th>XRD Variable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRDADMINPATH</td>
<td>Is the directory for <em>cmsd</em>-specific files and sockets.</td>
</tr>
<tr>
<td>XRDCALLCONFGN</td>
<td>The path to the configuration file.</td>
</tr>
<tr>
<td>XRDCMSCLUSTERID</td>
<td>The globally unique cluster identification for this host.</td>
</tr>
<tr>
<td>XRDEBUG</td>
<td>Set to one when the --d command line option is specified.</td>
</tr>
<tr>
<td>XRDHOST</td>
<td>The current host’s DNS name.</td>
</tr>
<tr>
<td>XRDMINSTANCE</td>
<td>Is the string of the form “execname instance@hostname”. Where execname is the executable’s name, instance is the name specified via --n or anon if no instance name was specified, and hostname is the current host’s DNS name.</td>
</tr>
<tr>
<td>XRDLOGDIR</td>
<td>Is the directory where log files are written.</td>
</tr>
<tr>
<td>XRDNAMEN</td>
<td>The name specified via --n or anon if no instance name was specified.</td>
</tr>
<tr>
<td>XRDPNAME</td>
<td>The executable’s name.</td>
</tr>
<tr>
<td>XRDSITE</td>
<td>The site name specified either via the --s command line option or the all.sitename directive.</td>
</tr>
</tbody>
</table>

If the standard *oss* plug-in is being used, the following additional environment variables are exported.

<table>
<thead>
<tr>
<th>OSS Variable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRDN2NLIB</td>
<td>The path and name of the name-to plug-in, if specified via the oss.namelib directive.</td>
</tr>
<tr>
<td>XRDRMTROOT</td>
<td>The local root path specified by the oss.remoteroot directive.</td>
</tr>
<tr>
<td>XRDLCROOT</td>
<td>The local root path specified by the oss.localroot directive.</td>
</tr>
</tbody>
</table>
2 Mandatory Configuration Directives

This section describes directives that are must be specified to configure the Cluster Management Service.

2.1 manager

```
all.manager [ meta | peer | proxy ] [ all | any ]

host[+]{:portspec | portspec} [ if conds ]

portspec: port[@iname][@sname]
```

Function

Specify the manager **cmsd** location.

Parameters

- **meta** Identifies the **cmsd** meta-managers that **cmsd** managers should subscribe to.
- **peer** Identifies the **cmsd** peer managers that **cmsd** managers should subscribe to as a peer manager.
- **proxy** Identifies the **cmsd** managers that **xrootd** servers with proxy roles (i.e., “proxy” or “proxy server”) should subscribe to.
- **all** Uses a load distribution algorithm to select an appropriate manager. See the section “Choosing all vs. any” for non-peer managers and the section “Peer Manager File Location” for peer managers to determine the best option for your cluster.
- **any** Uses a fail-over algorithm to select an appropriate manager. See the section “Choosing all vs. any” for non-peer managers and the section “Peer Manager File Location” for peer managers to determine the best option for your cluster.
- **host** The DNS name or IP address of the **cmsd** manager. If **host** ends with a plus sign (+), then the all hosts addresses associated with host are considered to be available managers.
port The TCP port number or service name at which the manager will accept connections. The port may be specified with an adjacent colon or space separation.

iname Associates the specified manager with an instance name (i.e. the name specified using the -n command line option). The iname is only used to support running multiple managers on the same physical host.

sname Places the specified manager into a group identified by an arbitrary 1- to 63-character name, typically the site name. By default, the name local is used. The sname is only used to support disjoint cluster configurations, discussed later.

conds The conditions that must exist for this directive to apply. Refer to the description of the if directive on how to specify conds.

Defaults
None; see the Notes for requirements. If you do not specify all or any, then any is assumed.

Notes
1) You must specify the “manager” directive for each xrootd given a manager role and for every cmsd given a server or supervisor role.
2) You must specify the “manager peer” directive for every cmsd given a peer or peer manager role.
3) You must specify “manager proxy” directive for each xrootd given a proxy or proxy server role.
4) This is a global directive and must be qualified by the “all” prefix.
5) All non-peer manager cmsd’s use the manager directive to establish a communications channel with each indicated manager.
6) You may specify up to 16 different managers.
7) If the manager host name ends with a plus, then all the IP addresses associated with host are treated as managers and every non-manager cmsd and xrootd subscribes to each one. This allows you to easily construct fault-tolerant configurations using DNS IP address aliases.
8) The host specifies the machine that is running cmsd in a manager role.
9) IP addresses may be specified in IPV4 format (i.e. “a.b.c.d”) or in IPV6 format (i.e. “[x:x:x:x:x]”).
10) Manager IP addresses are resolved once at start-up time and all specified managers should be registered in the DNS. The requirement is relaxed in dynamic DNS configurations (see the dyndns option of the xrd.network directive). For dynamic DNS configurations (e.g. Kubernetes), resolution occurs at run-time when needed.

11) Associating a manager with an instance name allows a manager to pick out its own specification based on the associated instance name. This is required when multiple managers for the same cluster run on the same physical host. Alternatively, you could run each manager in a separate container and assign a unique hostname to each one. Then it is immaterial whether or not the containers are running on the same physical host.

Example

```
all.manager beastmanager.slac.stanford.edu 1213
```
2.1.1 Choosing all vs. any for Normal Managers

When more than one manager is present the **all** and **any** options control how a manager is selected. Be aware that this section discusses these options for normal managers (i.e. not peer managers). The all and any options as they apply to peer managers are discussed in the next section.

In order to understand the **all** and **any** options you should be familiar on how **xrootd** and **cmsd** managers provide robustness. In the figure below we have three manager **xrootd-cmsd** pairs. The **xrootd** accepts file-oriented requests and asks the **cmsd** to resolve the files location. The **xrootd** client provides robustness by simply selecting at random some working **xrootd**. This distributes the load across all **xrootd** daemons. On the other hand, each **xrootd** daemon actually connects to all possible **cmsd** manager daemons and now has a choice of which working manager to use. The **all** and **any** options only affect how an **xrootd** daemon selects a **cmsd**.

When **all** is specified, the **cmsd** uses a hash of the target file name to determine which manager is to handle the file lookup request. This effectively distributes the load across all available managers. If one of the managers fails, it is temporarily replaced by another working manager until the failed manager becomes operational and the load can once again be equally distributed. The manager selection algorithm is effective even when multiple managers fail. Choose the **all** option if you expect a heavy file lookup load.

When **any** is specified, the **cmsd** designates one of the managers for all file lookup requests. If that manager fails, the next available working manager is used. When the failed manager becomes operational it is once again designated as the preferred manager. This option provides simplicity for debugging file location problems since only one manager is handling all file lookup requests and only one log usually needs to be consulted. Use the **any** option when you expect light loads. Consider using the **all** option if you see one of the **cmsd** using more than about 4% of the CPU or grow beyond 1 GB of memory.
2.1.2 Peer Manager File Location and Selection

The **manager** directive with the **peer** option identifies managers of other peered clusters. It is only used by servers that have a non-proxy manager or meta-manager role. Peer clusters are destinations of last resort. When a file cannot be found in the cluster and there is an eligible peer cluster that could potentially serve the file, the client is redirected to the peer cluster. Peer clusters are never searched for a file by another peer manager. In effect, they are independent clusters that may or may not have the file of interest.

A peer cluster can have its own set of peer clusters and generally peer relationships are reciprocal in nature. That is if manager A has peer B then B would naturally name A as its peer manager. When a client is redirected to a peer, the redirecting manager prohibits that peer from redirecting back to it. This avoids a redirection loop.

Peer selection is controlled by the **any** and **all** options. The default is **all** which means clients will be redirected to peers in the order they are listed. For instance, if two peers are listed as in order B and C then a client will always be redirected to B unless B is not available, in which case it will be redirected to C.

When **any** is specified on the first manager peer directive, then clients are redirected to peers in least recently used order. Unlisted peers subscribing to a manager receive the **any** option. If **all** is in effect, these peers are selected last.

Because peer clusters are never searched by a peer manager, locate requests directed to a peer manager do not, by default, list peers. In certain contexts, this may produce less than optimal results (e.g. **xrdcp** extreme copy mode). The **kXR_locate kXR_addpeers** option may be used to also display eligible peers. It is important to remember that these peers might not have the file in question and a manual search is needed to determine if they do. This automatically happens in recursive location requests but should be avoided for broad requests (e.g. directory listing) in order to minimize network traffic.

Displayed peers cannot be readily differentiated from local resources. However, it is possible to restrict locates to peers by prefixing the path with an equals sign ("="). The result indicates which peers need to be searched determine the file’s actual location.
2.1.3 Disjoint Cluster Configurations

Normally, the manager directive identifies all of the managers for a particular collection of servers, called a cluster. When you identify more than one manager, the members of the cluster assume that the managers are functionally identical (i.e. merely replicas setup for enhanced reliability). The figure on the left shows such a configuration. Here SM, a server, joins the two managers, M-A1 and M-A2, of Cluster A. Then SM becomes part of that uniform cluster. Thus, a request issued by one of the managers is automatically done relative to all of the managers. This provides cluster cohesion regardless of how many managers exist and the all.manager directive is the same for all members of cluster A.

For instance, if one of the managers of a cluster blacklists and redirects a member of the cluster, that member assumes that the redirect is to be taken relative to all of the managers. Hence, the member disconnects from all of the managers and connects to the nodes to which the member was redirected by one of the managers.

This mode of operation is correct as long as all of the managers are indeed replicas of each other. However, it is possible to construct a cluster whose members provide resources to two disjoint clusters, say A and B, as shown in the figure to the left. In this case, SM still needs to identify the managers of A and the managers of B. But in this case, they really are not replicas of each other. Indeed, managers of A are distinct from the managers of B. Treating all of them identically would quickly make such a configuration dysfunctional.
SM avoids such a conflict by using the *sname* qualification in the `all.manager` directive when defining the managers in its own configuration file. Here, managers in cluster A were qualified with `@A` while managers in cluster B were qualified with `@B`. This allows SM to treat these as two as unrelated managers yet provide services to both managers in a uniform way.

The ‘@’ suffix is an arbitrary name and is merely used to distinguish the managers. If you employ site naming (i.e. `xrd.sitename` directive) then the suffix should be the site name assigned to each cluster. This makes log file messages more descriptive; especially for such a complex cluster configuration.

Had SM been a manager, then the `all.manager` directive in its configuration file would identify the managers of A and B as meta-managers, as

```
all.manager meta hostname:port@sname
```

Finally, you can avoid listing each individual manager by manager by creating a DNS entry that is associated with two address records, one for each manager in the cluster. Doing this would allow you to simply enter the DNS alias for both managers indicating that the addresses should be automatically expanded as in

```
all.manager meta hostname:port+@sname
```
2.2 role

```
all.role rolename [ if conds ]

rolename: [meta | proxy] manager | [proxy] server |
[proxy] supervisor
```

**Function**
Designate the role the server is to have.

**Parameters**

`rolename`
The server’s role in the configuration. See the usage notes and the following section for an explanation of roles.

`conds` The conditions that must exist for this directive to apply. Refer to the description of the `if` directive on how to specify `conds`.

**Defaults**
```
all.role manager
```

**Notes**
1) This is a global directive and `must` be qualified by the “all” prefix.
2) Do not specify the `all.role` directive when configuring a stand-alone XRootD server. This directive specifies that the server is part of a cluster and that a local `cmsd` exists. Stand-alone servers, by definition, do not have a `cmsd`. Failure to ignore this will fill the log with error messages indicating that the local `cmsd` cannot be contacted.
3) A role of `manager` indicates that the `cmsd` is at the top-most level of the server hierarchy and is used to locate files.
4) A role of `server` indicates that `cmsd` is at the bottom-most level of the server hierarchy and is used by pure data servers to serve data files.
5) A role of `supervisor` indicates that the `cmsd` is at an intermediate-level of the server hierarchy and is used to bridge the top-most level and the bottom-most level.
6) A role of **meta manager** indicates that **cmsd** is to act as a manager and accept subscriptions from other managers. Meta managers allow you to federate administratively independent clusters.

7) A role of **proxy** indicates that the **xrootd** is at the top- *and* bottom-most level of the server hierarchy. When contacted, the **xrootd** acts like a **manager** to locate the target file. However, unlike a true redirector, the **xrootd** actually performs the requested operation as if it were a **server** acting in behalf of the client making the request.

8) Any **xrootd**’s designated as proxies may only communicate with **cmsd**’s that have also been designated as proxies.

9) The following table describes the effect each role has on an **xrootd** server and its corresponding **cmsd** server.

**Example**

```
all.role supervisor if sup*.slac.stanford.edu
```

### 2.2.1 Role Summary Table

<table>
<thead>
<tr>
<th>Role</th>
<th><strong>cmsd</strong></th>
<th><strong>xrootd</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>manager</strong></td>
<td>Provides a search service across one or more “server” or “supervisor” <strong>cmsd</strong>’s.</td>
<td>Logs into one or more <strong>cmsd</strong>’s, identified by the “manager” directive, and provides a redirection service</td>
</tr>
<tr>
<td><strong>server</strong></td>
<td>Subscribes to a “manager” <strong>cmsd</strong>, identified by the “manager” directive, in order to form a cluster <em>and</em> accepts logins from a local <strong>xrootd</strong>.</td>
<td>Logs into a local “server” <strong>cmsd</strong> and provides data from a locally accessible file system.</td>
</tr>
<tr>
<td><strong>supervisor</strong></td>
<td>Same as “server” <em>plus</em> provides a search service across one or more server or supervisor <strong>cmsd</strong>’s.</td>
<td>Logs into a local “supervisor” <strong>cmsd</strong> and provides a redirection service.</td>
</tr>
<tr>
<td><strong>meta manager</strong></td>
<td>Provides a search service across one or more “manager” <strong>cmsd</strong>’s.</td>
<td>Logs into one or more <strong>meta manager</strong> <strong>cmsd</strong>’s, identified by the “manager meta” directive, and provides a redirection service.</td>
</tr>
<tr>
<td>Role</td>
<td>cmsd</td>
<td>xrootd</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>proxy manager</td>
<td>Same as “manager” but only accepts cmsd’s and xrootd’s that have a “proxy” role (i.e., can only manage proxies).</td>
<td>Same as manager role except that the manager cmsd’s must also have a proxy role.</td>
</tr>
<tr>
<td>proxy server</td>
<td>Same as “server” except that managers and the local xrootd must also have a proxy role.</td>
<td>Same as “proxy and logs into a local “proxy server” cmsd to be part of a cluster.</td>
</tr>
<tr>
<td>proxy supervisor</td>
<td>Same as “supervisor” but only allows proxy cmsd subscriptions (i.e., can only manage proxies).</td>
<td>Logs into a local “proxy supervisor” cmsd and provides a redirection service.</td>
</tr>
</tbody>
</table>
3 Common Configuration Directives

3.1 allow

```
cms.allow { host | netgroup } name
```

Function
Restrict the hosts that can subscribe to the manager `cmsd`.

Parameters
host name
The DNS host name or IP address allowed to subscribe to the `cmsd`. Substitute for `name` a host name or address. The host name may contain a single asterisk anywhere in the name. This lets you allow a range of hosts should the names follow a regular pattern. IP addresses may be specified in IPV4 format (i.e. “a.b.c.d”) or in IPV6 format (i.e. “[x:x:x:x:x]”).

netgroup name
The NIS netgroup allowed to subscribe to the `cmsd`. Substitute for `name` a valid NIS netgroup. Only hosts that are members of the specified netgroup are allowed to subscribe to the `cmsd`.

Defaults
None. If `allow` is not specified, any host is allowed to subscribe.

Notes
1) This directive is only used by manager-mode `cmsd`’s.
2) You may specify any number of hosts and netgroups. Any host matching a specified name or is a member of a specified netgroup is allowed to subscribe to the `cmsd`.
3) **Warning!** Using hostname based security relies on the security of the DNS server and the inability of other hosts spoofing and successfully using the “allowed” IP addresses. The two security assumptions have severe limitations.
4) Use strong authentication to provide a more robust security framework. Refer to the `seclib` directive for more information.
Example

cms.allow host kandata*.slac.stanford.edu
3.2 `defaults` (an oss directive)

```
oss.defaults options

         {forcero | readonly | r/o | r/w | [not]writable}
         {inplace | outplace} {local | global | globalro}
         {[no]mig | [not]migratable} {no|mkeep [no]mlock
```

**Function**
Specify default file processing options.

**Parameters**

<table>
<thead>
<tr>
<th>Option</th>
<th>Disabled/Enabled Function</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>forcero</td>
<td>Convert all file open requests to read-only access (cmsd &amp; oss).</td>
<td>writable</td>
</tr>
<tr>
<td>local</td>
<td>Do not export this path via the cluster manager (cmsd only).</td>
<td>global</td>
</tr>
<tr>
<td>global</td>
<td>Export this path via the cluster manager (cmsd only).</td>
<td>global</td>
</tr>
<tr>
<td>globalro</td>
<td>Export this path via the cluster manager as read-only (cmsd only).</td>
<td>global</td>
</tr>
<tr>
<td>readonly</td>
<td>Files may only be opened for read access (cmsd &amp; oss).</td>
<td>writable</td>
</tr>
<tr>
<td>r/o</td>
<td>Path is writable (cmsd &amp; oss)</td>
<td>writable</td>
</tr>
<tr>
<td>r/w</td>
<td>Path is writable (cmsd &amp; oss)</td>
<td>writable</td>
</tr>
<tr>
<td>[no]stage</td>
<td>[Do not] stage a file from a remote storage system should it not exist in the local file system at open time.</td>
<td>nostage</td>
</tr>
<tr>
<td>[not]writable</td>
<td>Path is [not] writable (cmsd &amp; oss).</td>
<td>writable</td>
</tr>
</tbody>
</table>

---

2 Only cmsd-related options are shown in the table. Other options are specific or xrootd or the oss component. Consult xrootd and ofs/oss references for details on unlisted options.
Notes

1) This directive is identical to the `oss.defaults` directive and establishes the defaults for the `export` directive. This allows you to keep a single configuration file for cms and oss components.
2) Directive options may be applied to selected paths using the `export` directive. This allows you to selectively over-ride the default.
3) The `defaults` directive _should_ be specified prior to any `export` directives.

Notes on `forcero` and `readonly`

1) The `forcero` and `readonly` options declare any files prefixed by the path to be non-writable. The `cmsd` excludes all servers declaring the prefix as non-writable when looking for a file that is to be modified or created.
2) The `mlock`, `mkeep`, and `mmap` options cause a path to have the `forcero` attribute.

Notes on `local`, `global`, and `globalro`

1) The `local` option prevents the applicable paths to be seen by the manager `cmsd`; making them globally inaccessible via the redirector.
2) The `global` option makes a path eligible to be used by the manager `cmsd` and associated redirector. This is the default.
3) The `globalro` option makes a path eligible to be used by the manager `cmsd` and associated redirector in `readonly` mode; regardless of how it is actually declared for the server. This allows you to export local `writable` paths as global `readonly` paths.

Notes on `[no]stage`

1) When `stage` is in effect, files are dynamically staged from a remote storage system to local file space when opened, if the file is not already locally on disk. The `cmsd` selects servers that can stage the file should no other server have the file or if otherwise eligible servers are overloaded or unavailable.
2) When `nostage` is in effect, the server claims that the files must exist on disk in order to be accessed.
3) The `nostage` and `stage` directives may be applied to selected paths using the `path` directive.

Example

`oss.defaults stage forcero`
3.3 dfs

```plaintext
cms.dfs [limit [central] [=] rate]
    [lookup {central | distrib}] [mdhold mdtm]
    [qmax qmax] [redirect {immed | verify}]
    [retries rmax]
```

Function
Configure distributed file system handling.

Parameters

**limit** Establish limits on meta-manager requests. The limit is applied in the manager node when `central` is specified. Otherwise, the limit is applied where file systems look-ups occur (see `lookup`). The `rate` specifies the number of look-ups per second allowed. When `rate` is preceded by an equals sign (=), look-ups are metered to occur exactly at the specified `rate`. Otherwise, the system uses a median-average algorithm. See the notes on how these algorithms differ. By default, no limit is applied and is equivalent to specifying zero or a value greater than 1000 for `rate`.

**lookup**
Specifies where file existence checks are to be performed. By default, they are performed on data server nodes (i.e. `distrib`). If the manager node has access to the distributed file system, file existence can be checked by the manager if `central` is specified. See the notes on the pros and cons of using `central` vs. `distrib` look-ups.

**mdhold**
Instructs data servers to keep track of missing directories for `mdtm` time. The `mdtm` may be suffixed by s (the default), m, or h to indicate seconds, minutes, and hours, respectively. When a look-up for a non-existent file occurs, the data server automatically looks-up its parent directory and records whether or not it exists. If the directory does not exist, the fact is remembered for `mdtm`. Otherwise, the fact that the directory exists is remembered for `mdtm*10`. Subsequent look-ups for files in missing directories will
immediately fail. See the notes on appropriate values. The default is zero which turns this optimization off.

qmax  Specifies the maximum number of look-ups that can be queued for processing. When qmax is reached, the oldest unprocessed look-ups are deleted and no look-up is performed; effectively returning a “file does not exist” response. The qmax is ignored if no rate limit applies. The minimum value is 1. The default value is rate*2.5. See the notes on how this interacts with rate.

redirect  Specifies how the manager is to handle file look-ups by clients. When immed is specified, no look-up is performed and the client is immediately directed to the most suitable data server where the client re-drives the look-up. This is the default for proxy managers. When verify is specified, the manager first determines whether or not the file exists (either locally or via a server query, as specified by lookup) and if the file exists, the client is directed to a suitable server. This is the default for non-proxy managers. See the usage notes on how this option affects performance.

retries  Specifies how many servers a client may exclude when reselecting a server in the DFS cluster due to an error. The default is 2.

Defaults
Proxy manager:
cms.dfs limit 0 lookup distrib mdhold 0 redirect immed retries 2

Otherwise:
cms.dfs limit 0 lookup distrib mdhold 0 redirect verify retries 2

Notes
1) When you specify the dfs directive, the cmsd optimizes file processing to avoid duplicate requests for the file. A distributed file system is essentially a shared-everything system and if one data server has access to a file, all data servers have access to the same file. Examples of distributed file systems are: dCache, GPFS, HDFS, Lustre; and xrootd proxy servers.
2) By default, the cmsd assumes a shared-nothing system where each data server has its own independent file system. In order to determine who has a file, all data servers are necessarily interrogated.
3) As in shared-nothing systems, the `cmsd` still honors the exported paths declared by servers in a shared-everything system. So, while each server has access to all files in the file system you may logically restrict access by exporting different paths from each server.

4) The `limit` parameter only applies to meta-manager requests and provides you with the option of limiting the impact of external queries. If the `limit` is set too low or is set much higher than the ability of the underlying file system to handle look-ups (i.e., `stat()`), files will appear to be non-existent to the meta-manager at the queried node.

5) When `limit` is specified, the default is to use a median-average algorithm to limit look-ups. This algorithm allows for brief bursts of activity before applying deterministic pacing. This kind of algorithm is much more responsive and requires less CPU time. However, it can subject the underlying file system with up to 50% of the allowed look-ups in a very brief period of time. The alternative is to pace the look-ups at a deterministic rate. While this is not as responsive and requires more CPU time, it guarantees a predictable file system load.

6) The `limit rate` is directly affected by the `lookup` parameter. When look-ups are done by managers (i.e., `lookup central`) the actual rate is equal to the specified value times the number of managers configured to talk to a meta-manager. When look-ups are done by data servers (i.e., `lookup distrib`, the default) the actual rate is equal to the specified value times the number of data servers whose manager talks to a meta-manager. In either case, the number of meta-managers does not affect the rate.

7) In general, it is rare that a limit needs to be specified and is normally not recommended.

8) The `lookup` parameter controls where file look-ups occur. The default is to spread look-ups across all of the data servers. This greatly increases scalability at the cost of increased latency. If the look-up request rate is relatively low or the underlying file system can process a look-up in less than a few milliseconds, then specifying `central` can reduce the latency while maintaining reasonable scalability. The underlying distributed file system must be available to the manager when central look-ups are enabled. The choice of look-up impacts the specified `limit`, if any.
9) The **mdhold** parameter allows you to reduce the overhead when looking up missing files when look-ups are done for files in a missing directory. The **mdhold** parameter control how long the **cmsd** is to remember that a directory is missing. Future look-ups in a missing directory immediately fail without actually checking the underlying file system during this time. Since the **cmsd** has no way of knowing if a directory was actually created during this time, the hold time should be set to a small value and should not be specified at all if directories are actively created for files likely to be looked up either by the meta-manager or the manager. Excessively long hold values will likely result in files being incorrectly tagged as missing.

10) The **mdhold** processing occurs where look-ups are preformed (see the **lookup** parameter). The **limit** parameter does not apply to directory look-up requests.

11) The **mdhold** option is not compatible with **http[s]** and may cause **http** to report an error when creating a directory.

12) The **qmax** parameter controls the maximum number of outstanding look-up requests. It is only meaningful when a limit is in effect. Look-up queues may occur when the limit is set too low or when the actual file system look-up rate is lower than the specified limit. When the queue limit is reached, the oldest requests are discarded and the file is deemed missing for those look-up requests.

13) The **redirect** parameter can also be used to optimize look-ups. In **immed** mode (the default for proxy managers), the manager immediately redirects clients to a suitable data server without any file look-up at all. The server is responsible for doing the look-up. In **verify** mode (the default for non-proxy managers), the manager performs the look-up to ensure that a selected data server will in fact be able to process the client’s request. The choice of mode should be done in the context of how clients reach a manager node. If most of the activity is local to the cluster then **verify** mode is usually better. If most of the activity is generated by meta-manager redirects then **immed** mode is usually better.

14) The **retries** option provides a limit on how many times a client may reselect a server. The notion is that since all servers in a **DFS** cluster are the same, an error encountered on one would also occur on any other **DFS** server. The default allows for two tries before the error is considered permanent. This avoids needlessly redirecting clients to other servers.

15) Do not confuse the **retries** option with the option **maxretries** option in the **sched** directive. The **sched maxretries** applies a similar limit only to regular clusters.
16) The `dfs` directive is meant to be used for clusters exporting a distributed file system or for proxy *non-caching* clusters. Other uses are not supported and may produce unwanted effects.

**Example**

```
cms.dfs lookup central mdhold 1m
```
3.4 export

```
all.export  path [ xoptions ] [ options ]
```

Function
Specify processing options for any path matching the specified path prefix.

Parameters
path The path prefix to which the specified options apply. If no options are specified, the current defaults are used.

xoptions
xrootd options to apply to any path whose prefix matches path. See the export directive described in the xrd/xrootd configuration reference.

options
oss and cmsd-specific (i.e., local, global, and globalro) options to apply to any path whose prefix matches path. Refer to the oss.defaults directive for a detailed explanation of these options.

Defaults
All paths are processed according to the default options in effect at the time the path directive is encountered. Defaults are set using the defaults directive.

Notes
1) Any number of export directives may be specified. They are cumulative and are checked in decreasing length order (i.e., most-specific to least specific).
2) The export directive is usually defined when configuring xrootd and the oss component. Additional cmsd-specific options may also be included.
3) The export directive is used by xrootd and cmsd to determine which paths are valid for incoming client requests.
4) The export directive is used by oss component to enforce desired processing attributes.
Example

```
all.export /xrd/files/staged mig nodread rcreate
```
3.5 **localroot** (an oss directive)

```
oss.localroot path
```

**Function**

Specifies where the local file system name space is actually rooted.

**Parameters**

*path*  The path to be pre-pended to any local path specified by a client request.

**Defaults**

None. Paths are used locally as specified.

**Notes**

1) The **localroot** parameter allows you to keep the external namespace consistent even when you move the associated file system from one location to another. Say that a file system is mounted at `/xrd`. This means that all file paths start with `/xrd`. If now you needed to mount the file system at `/usr/xrd` then by specifying

```
oss.localroot /usr
```

the external view of the file system would remain the same since **oss** will automatically prefix all paths with `/usr` and use the new mount point.

2) The **cmsd** honors the **oss localroot** directive. This allows you to use a single configuration file for the **cms** and **oss** components.

**Example**

```
oss.localroot /usr
```
3.6 perf

\texttt{cms.perf [xrootd] args}

\textit{args:} \ [ \texttt{int\ \textit{time}} \] \ [ \texttt{lib\ \textit{path} [\textit{parms}] | \texttt{pgm\ \textit{prog}} ]

\textbf{Function}

Specify how load is computed and reported.

\textbf{Parameters}

\texttt{xrootd}

The directive only applies to the \texttt{cms} client running in an \texttt{xrootd} server. This option allows reporting load from the \texttt{xrootd} server as opposed to the \texttt{cmsd}.

\texttt{int\ \textit{time}}

When \texttt{pgm} is specified; the estimated \textit{time} between load reports as computed by \textit{prog}. When \texttt{lib} is specified, the rate at which performance information is to be retrieved from the plug-in. The time may be suffixed by \texttt{s} (the default), \texttt{m}, or \texttt{h} to indicate seconds, minutes, and hours, respectively.

\texttt{lib\ \textit{path} [\textit{parms}]}

The load is computed by a plug-in residing in the shared library identified by path. The plug-in reports it when asked or pushes the information to the server. The parms, if any, are passed to the plug-in when it is loaded. The \texttt{lib} parameter must be the last parameter on the line.

\texttt{pgm\ \textit{prog}}

The program that computes the machine load and write the information to standard out. The \texttt{pgm} and \texttt{xrootd} parameters are mutually exclusive. The \texttt{pgm} parameter must be the last parameter on the line.

\textbf{Defaults}

\texttt{cms.perf int 3m}
Notes

1) This directive is only used in servers with a role of server.
2) There is no default value for the program or library and load information cannot be collected and reported unless a load collector exists. A sample program, cms_MonPerf, is supplied for this purpose. This program uses the rperf command, among others, to calculate the cpu, i/o, and various other load levels.
3) The specified program is started by the server-mode cmsd at startup time. It is automatically restarted after two failures to report a load within the specified interval.
4) The xrootd argument is meant for custom installations where the load is better computed in the data server than an external source. If you decide to use this option you should not include a directive that does not contain the xrootd option.
5) The specified program must write 5 white-space separated numbers to standard out. The last number must be terminated by a new-line character ("\n"). Each number must be normalized to 100, with 0 indicating no load and 100 indicating saturation. The numbers are in the order:
   1. system load
   2. cpu utilization
   3. memory utilization
   4. paging load, and
   5. network utilization.
6) Performance can also be supplied by a plug-in. Refer to the “include” file XrdCmsPerfMon.hh for details.

Example

cms.perf int 5m pgm /usr/etc/ooss/olb_MonPerf 300
### 3.7 prep

**Function**

Specify how offline file preparation is done.

**Parameters**

- **echo**  Writes to the log all of the files found in the external in-preparation queue whenever a reset occurs.

- **reset cnt**
  
  The maximum number of scrubs of the in-preparation queue that can be done before the contents of the queue are recomputed. The default is three (3).

- **scrub time**
  
  The `time` between scrubs of the in-preparation queue. The `time` may be suffixed by `s` (the default), `m`, or `h` to indicate seconds, minutes, and hours, respectively. The default is 20 minutes.

- **ifpgm ifprog**
  
  If specified, `ifprog` replaces the default built-in prepare mechanism and becomes the interface that adds, removes, and lists preparation queue files. The following section describes the input, output, and calling conventions that `ifprog` must have. The `ifpgm` parameter must be the last parameter on the line. Any parameters after `ifprog` are passed to the program via the argument list. Quoted values must be avoided as they are not correctly passed.

**Defaults**

None. Preparation queue handling is normally disabled.

**Notes**

1) This directive is only used by server- and manager-mode `cmsd`'s.

2) The default prepare mechanism relies on the File Residency Manager’s `frm_xfragent`. You must configure and run `frm_xfrd` to successfully implement the default prepare mechanism.
3) Each **cmsd** that can stage files is also capable of preparing files to be online prior to their active use. This is done through the prepare protocol. The mechanism that is actually used to bring files to local disk is the responsibility of the external infrastructure.

4) The **prep** directive enables and, optionally, describes the interface to that infrastructure. If you do not specify the **prep** directive, even with no arguments, file preparation is disabled.

**Example**
```
cms.prep scrub 10m ifpgm /opt/xrd/bin/prepare_xrdsync
```
3.7.1 Optional Prepare Interface Program Requirements

Most installation chose the default mechanism to route file preparation requests. This employs the File Residency Manager along with `frm_afragent` and `frm_xfrd`. Refer to the File Residency Manager Reference for full details. If you have special needs, you can over-ride the built-in default by specifying an `ifprog` (see previous section). The requirements of this program are:

1) The `ifprog` is used to add, remove, and list reparation queue files. When specified, it is started at initialization time and is expected to run continuously, and is automatically restarted should it fail. Parameters are sent via standard in, one request for each new line terminated record. Except for the “list” (i.e., `?`) request, the program should not write any output to standard out. Output to standard error is included in the `cmsd` log file.

2) When the `cmsd` needs to know the exact contents of the preparation queue (e.g., files waiting to be brought to local disk) it sends a single question. Refer to the default prepare query message for the exact response requirements.

3) The format of the messages sent to the program is described under the `prepmsg` directive. To the prepare query message description for the required response.

4) If prepare notification is requested, the command should adhere to the following message format:

   Successful: \texttt{\textbf{ready} requestid msg path}
   
   Unsuccessful: \texttt{\textbf{unprep} requested msg path}

   \texttt{requestid} is the request identifier associated with the completed request. \texttt{msg} is the text that followed the notification url (see the `prepmsg` directive). This text must be sent without inspection. \texttt{path} is the logical name of the file that successfully prepared or whose preparation failed.

5) Because file preparation is done on a best-effort philosophy, the preparation program is free to honor (or not) the requests in any way. Currently, the `cmsd` does not check the return status of the program nor expects any error output (e.g., messages).
3.7.1.1 Default Prepare Request Message (prepmsg)

The default message that is sent to the prep ifpgm’s stdin when a prepare operation is required has the following format:

\[+[\text{traceid}] \text{requestid} \text{npath} \text{prty} \text{mode} \text{path} [\text{path} [\ldots]]\]

Where:

- **traceid**: The unauthenticated identifier associated with the client making the request. The traceid is automatically included when communicating with the File Residency Manager (frm).

- **requestid**: The request identifier that can be used to group this request into a unique set of requests. The requestid is globally unique.

- **npath**: The notification path to be used to indicate how the request complete. This field may contain:
  - no notification is to be sent.
  - file:///path send msg via local named pipe named path
  - mailto://user send e-mail to user
  - tcp://rhost:port/msg send msg via tcp to rhost:port
  - udp://rhost:port/msg send msg via udp to rhost:port

- **prty**: The request priority: 0, the lowest, to 2, the highest.

- **mode**: The processing mode and may contain a combination of the following letters:
  - f send fail notice (not affected by q flag)
  - n send success notice
  - q suppress default failure notice (i.e., quiet)
  - r file is expected to be only read
  - w allow the file to be modified

- **path**: The absolute logical name of the file to be prepared. If more than one path is specified, each path is separated by a blank.

---

3 This message may be specified by using the stagemsg directive.
Notes

1) You can change the format of a prepare request message with the prepmsg directive. However, you cannot use the supplied `frm_pstga` and `mps_prep` commands unless you use the default format.

3.7.1.2 The Prepare Cancel Message

The following message is sent to the `prep ifpgm`'s stdin to cancel a stage operation:

```
- requestid
```

Where:
`requestid` The request identifier used in a previous prepare request. All entries with this `requestid` should be removed.

Notes

1) You cannot change the format of a prepare cancel request message.

3.7.1.3 The Prepare Query Message

The following message is sent to the `prep ifpgm`'s stdin to cancel a stage operation:

```
?
```

Notes

1) The `ifprog` should respond with a list of new-line separated absolute paths associated with queued requests.
2) You cannot change the format of a prepare query request message.

---

4 `mps_prep` along with `mps_PreStage` and `mps_Stage` are deprecated. The `frm_xfrd` should be used instead.
3.8 sched

cms.sched parms

 parms: [ affinity [default] {none|weak|strong|strict} ]

 [ affpath {all | first n | last n} ]

 [ cpu pcpu ] [ io pio ] [ mem pmem ] [ pag ppag ]

 [ runq prunq ] [ space putl ] [ fuzz fnum ]

 [ gsdlit gsdp ] [ gshr gsp ] [ maxload mload ]

 [ maxretries mrt[@host:port] ]

 [ nomultisrc[@host:port] ] [ refreset sec ]

Function

Specify the parameters for the load balancing scheduling algorithm.

Parameters

affinity [default] {none|weak|strong|strict}

File affinity policy that the redirector should use when selecting a server.

default

the specified affinity is merely a default and a client may select
an alternate affinity using the “cms.aff” CGI tag (see the notes
for details). Without default the specified affinity is mandatory.

none

files have no affinity and servers should be selected to distribute
requests across all servers. This is the default and uses load
information if it has been configured.

weak

files have affinity to the longest-lived server however when the
location of the file is not known, the client is directed to the first
server that declares it has the file. Otherwise, the longest lived
server that has the file is always used. Load information is used
if it has been configured.

strong

files have affinity to the longest-lived server and when the
location of the file is not known, the client is delayed until all
locations of the file are known. Only then is the client redirected.
longest lived server that has the file is always used. Load information is used if it has been configured.

**strict** same as strong but load information is never used even when it is available. This guarantees that the longest-lived server is always chosen regardless of its load.

**affpath** \{all | first \(n\) | last \(n\)\}

File affinity path policy that the redirector should use when selecting a server. Affinity is determined by the file path or some portion of it, as follows:

- **all** the full path, this is the default.
- **first** the first \(n\) components of the path.
- **last** the last \(n\) components of the path.

**cpu** \(tcpu\)

The percentage of cpu load to be used to compute the overall load of a server.

**fuzz** \(fnum\)

The percentage difference two overall load values must have before they are considered different. A value of 100 suppresses the use of load in any scheduling decisions.

**gsdflt** \(gsdp\)

The default share the meta-manager should use in the absence of a manager-specific value. The default is 100. See the notes for more information.

**gshr** \(gsp\)

The maximum percentage of meta-manager requests that should be directed to this manager (i.e. the global share). The default is 100. See the notes for more information.

**io** \(pio\)

The percentage of io load to be used to compute the overall load of a server.

**maxload** \(mload\)

The maximum overall load a server may have. Servers whose overall load is greater than \(mload\) are not scheduled.
**maxretries mrt**

The maximum number of times a client can request an alternate server due to errors or to increase bandwidth to a file. By default, there is no limit. When mrt is suffixed by @host:port then the client is redirected to the specified host and port once the limit is exceeded. Specify a value between 0 and MAXINT. This option is meant to be used for disk caching proxy servers. See the notes for more details.

**mem tmem**

The percentage of memory load to be used to compute the overall load of a server.

**nomultisrc**

When specified, it disallows the client to request an alternate server to increase bandwidth to a file. By default, the client is allowed to do so. When the options is suffixed by @host:port then the client is redirected to the specified host and port is any attempt is made to get an alternate server on order to increase network bandwidth. This option is meant to be used for disk caching proxy servers. See the notes for more details.

**pag tpag**

The percentage of paging load to be used to compute the overall load of a server.

**refreset sec**

The number of seconds between server reference count resets. The time may be suffixed by s (the default), m, or h to indicate seconds, minutes, and hours, respectively.

**runq trunq**

The percentage of runq load to be used to compute the overall load of a server.

**space putil**

The percentage of space utilization to be used to compute the overall load of a server when selecting a server to stage or create a file.
Defaults

cms.sched affinity none affpath full

cms.sched cpu 0 io 0 mem 0 pag 0 runq 0 space 0 fuzz 20

cms.sched gshr 100 refreset 3600

Notes
1) This directive is only used by cmsd's with manager and supervisor roles.
2) The load-balancing algorithm chooses from all available servers the server whose computed overall load is smallest. When two servers have the same load, as determined by fuzz, the affinity option controls the selection (e.g. affinity none chooses the least selected server).
3) Other factors apply in selecting a server. For instance, whether or not the server has the requested file on disk, whether the server is allowed to dynamically stage a file, whether the server has sufficient disk space, etc.
4) The sum of pctcpu, pctoi, pctmem, pctpag, and pctrunq should be equal to 100.
5) If the sum of pctcpu, pctoi, pctmem, pctpag, and pctrunq is equal to zero, or if fuzz is 100, servers are selected by the affinity option (e.g. affinity none performs round-robin selection).
6) Mode scheduling is also forced when performance monitoring is disabled (see the ping usage directive).
7) Round-robin selection, with or without load information, is accomplished by using an internal reference counter in order to equalize the selection process. Since this counter may drift due to external anomalies encountered during scheduling, it is periodically reset. The refreset parameter controls the minimal reset frequency. However, the counter is only reset if sufficient selection activity occurred.
8) The gshr option allows you to set the maximum relative share of requests that a meta-manager subscriber wishes to accept from a meta-manager. Since the percentage is relative its effect is determined by the relative shares of other subscribers to the meta-manager. For instance, if all subscribers indicate the same share then this is equivalent to a share of 100 from the perspective of any individual subscriber. Hence, for global shares to be useful requires some amount of co-ordination between participating subscribers.
9) The global share is used by the meta-manager to select a subscriber only when a choice of subscribers exists (i.e., more than one subscriber has a requested file). In such a case, the meta-manager selects a subscriber so as not to exceed any individual subscriber's relative share of requests.
10) A subscriber’s share may be temporarily reduced if the subscriber is repeatedly selected because it is the only one which has a requested file.

11) The `gsdflt` option allows you to specify a default share (e.g. 50). This allows you to treat most subscribers the same and only differentiate those that are exceptions by giving them higher or lower shares than the normal default share.

12) The `gshr` and `gsdflt` options only apply to interactions with a meta-manager.

13) When the `default` is specified in the `affinity` option, then a client may choose a different affinity using the `cms.aff` CGI tag as follows:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Corresponding affinity</th>
<th>Tag</th>
<th>Corresponding affinity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cms.aff=n</code></td>
<td>none</td>
<td><code>cms.aff=s</code></td>
<td>strong</td>
</tr>
<tr>
<td><code>cms.aff=w</code></td>
<td>weak</td>
<td><code>cms.aff=S</code></td>
<td>strict</td>
</tr>
</tbody>
</table>
14) The **maxretries** and **nomultisrc** options are meant to be used for disk caching proxy server clusters to control the number of copies that may be created in the overall cluster. Using it for other purposes is not supported and may produce unwanted effects.

15) Do not confuse the **maxretries** option with the similar **retries** option in the **dfs directive**. They are orthogonal. While the effects are similar, the **sched maxretries** option is only applied to regular clusters while the **dfs retries** option is applied to **dfs** type clusters.

Example

```plaintext
cms.sched cpu 50 io 50
```
3.9 seclib

```
cms.seclib path
or
all.seclib path
```

**Function**

Specify the location of the security interface layer.

**Parameters**

- `path`: The absolute path to the shared library that contains an implementation of the Security (sec) interface that `cmsd` is to use for strong authentication.

**Defaults**

Strong authentication is disabled unless `seclib` is specified.

**Notes**

1) The `sec` interface allows you to provide an arbitrary authentication implementation (e.g., Kerberos, GSI, etc).
2) A `sec` implementation requires that compatible interface libraries be used on the server and client sides of the connection.
3) Refer to `XrdSecEntity.hh` and `XrdSecInterface.hh` for guideline on how to write a `sec` interface.
4) If you are using a common configuration file for all components (e.g., `xrootd` and `cmsd`) with security enabled; consider the following points.
   a. If the same security library is used for `xrootd` and `cmsd`, specify `all.seclib` to avoid having to specify the `seclib` directive twice.
   b. If a different set of protocols is being used for `xrootd` vs. `cmsd`, bracket the differences with an “if exec” construct. For instance,
      ```
      if exec cmsd
      security directives for cmsd
      else
      security directives for xrootd
      fi
      ```

**Example**

```
cms.seclib /opt/xrootd/lib/libXrdSec.so
```
3.10 space

\[ \text{cms.space } [\text{linger num}] [\text{recalc sec}] \]
\[ [\text{[min]} [\text{min%}] \text{min[k|m|g|t]}] [\text{[hwm%]} \text{hwm[k|m|g|t]}] \]

Function
Specify how servers are selected for file creation.

Parameters
linger num
The number of times a server may be reselected without an intervening server being selected for allocation. The default is zero (0).

recalc sec
The number of seconds between free space recalculations. The time may be suffixed by s (the default), m, or h to indicate seconds, minutes, and hours, respectively.

min%
The minimum amount of free space, as a percentage of the largest partition, a server must have in order for it to be selected. If the percentage is less than the min byte value, the min value is used.

min
The minimum amount of free space a server must have in order for it to be selected. You may suffix the byte quantity by k, m, g, or t to indicate kilobyte, megabytes, gigabytes, or terabytes, respectively.

hwm%
The minimum amount of free space, as a percentage of the largest partition, a server must have in order for it to be selected after free space has fallen below min. If the percentage is less than the hwm byte value, the hwm value is used.

hwm
The minimum amount of free space a server must have in order for it to be selected after free space has fallen below min. You may suffix the byte quantity by k, m, g, or t to indicate kilobyte, megabytes, gigabytes, or terabytes, respectively.
Clustering Configuration

Directives

Defaults

cms.space linger 0 recalc 15 min 2% 10g 5% 11g

Notes

1) This directive is only used by manager-and server mode cmsd’s.
2) The space values are used during server selection when either a file is
   opened in create mode or when a file must be dynamically staged.

Example

cms.space min 2g 10g
3.11 **space** (an oss directive)

```plaintext
oss.space  group  { path | ppfx* }
```

**Function**
Specify the location of one or more disk partitions.

**Parameters**
- **group**: The arbitrary name for the disk partition. Specify a 1- to 63-character name. While the name is required, the **cmsd** does not use it for any purpose.
- **path**: The absolute path at which the disk partition is mounted.
- **ppfx***: All directory entries that start with `ppfx` in the containing directory are to be used as disk partitions.

**Defaults**
None.

**Notes**
1) This directive is identical to the **oss.space** and, now deprecated, **oss.cache** directives. This allows you to keep a single configuration file for **cms** and **oss** components.
2) In order to redirect staging operations and file creations, the manager **cmsd** must know how much space is available on each server.
3) If the **xrootd** server is running a partitioned file system (i.e., files are allocated via symbolic links to one of many possible file system partitions) then specify each file system partition.
4) The path may end in an asterisk, indicating that all entries in the parent directory that start with the specified prefix are to be used as a file system partition. This is useful when partition mount points have regular names (e.g., `/data/space01`, `/data/space02`, etc.).
5) If the **cmsd** does not find any **space** directives, it infers the file systems to be used using the **export** directive.

**Example**
```plaintext
oss.space  public  /xrootd/space01
```
4 Esoteric Configuration Directives

This section describes directives that are normally not specified. You may wish to review these directives to be familiar with additional configuration options that are available.

4.1 altds

```
cms.altds xroot port [[no]monitor]
```

**Function**
Specify an alternate data server to pair with a cmsd server.

**Parameters**

- **port** Is the port number used by the alternate data server to service data requests using xroot protocol. The alternate data server must reside on the same node as the cmsd.

- **[no]monitor**
  The option specifies whether or not the cmsd server should monitor the availability of the alternate data server. The default is **monitor**.
  Specifying **nomonitor** makes the cmsd assume that the alternate data server is always available.

**Defaults**
None. The cmsd server assumes it is paired with a standard xrootd server.

**Notes**

1) The altds directive allows you to pair a cmsd configured for a server role with a non-standard data server using xroot protocol to supply data on the node where the cmsd is running. Client requests for data available on the node are automatically redirected to the alternate data server.

2) When **monitor** is in effect, the cmsd considers the alternate data server available as long as it is able to maintain an unauthenticated login session with the alternate data server.

**Example**
```
cms.altds xroot 2094
```
4.2 blacklist

```
cms.blacklist [check sec] [path]
```

**Function**
Black list one or more nodes.

**Parameters**
- **sec**: is the amount of time between checks whether or not the blacklist file has been changed. When a change is detected, the file is reprocessed and the blacklist updated. The time may be suffixed by s (the default), m, or h to indicate seconds, minutes, and hours, respectively. The default is 10 minutes (i.e. 10m) and may not be less than one minute.

- **path**: is the absolute path of the blacklist file. The default is name of the blacklist file is “cms.blacklist” which is assumed to exist in the configuration file directory.

**Defaults**
```
cms.blacklist check 10m configdirpath/cms.blacklist
```

**Notes**
1) Blacklisting is not applied unless the cms.blacklist directive is specified. You need not specify any options if the defaults are acceptable.
2) If the configuration file contains a cms.blacklist directive as well as a cms.whitelist directive, the last such directive applies.
3) Refer to the following major section on how to code a blacklist file.
4) The cms.blacklist directive only applies to nodes with a manager or meta-manager role.
5) Blacklisted nodes are prohibited from logging in. When a node’s login fails because it is blacklisted and is not redirected, the login is retried every minute until it succeeds or fails for another reason.
6) Black-listed may be redirected to another cluster. If this occurs, then no login retries are attempted at the redirecting host.
7) Redirection is only supported for CMS clients at version 4.2 or above. Clients below this version are effectively blacklisted and not redirected.
8) Nodes that are already logged in and found to be blacklisted and not redirected are disconnected and prohibited from logging in.

9) Nodes that are already logged in and found to be blacklisted and redirected are asked to disconnect and retry the login; which causes a redirect. If the node does not disconnect within the ping interval, it is forcibly disconnected.

10) To remove all hosts from the blacklist, simply remove the file.

11) If the blacklist file is not present, no controls are applied (i.e. all connections are allowed to login).

12) If the blacklist file is present but contains a syntax error or cannot be read, the current black is not changed.

Example

cms.blacklist /var/run/cms.blacklist
4.3 cidtag

```
cms.cidtag tag
```

**Function**
Specify the tag for the internally generated cluster identifier.

**Parameters**
tag  
a 1- to 16-character token. The token is added to the cluster identification string.

**Defaults**
None.

**Notes**
1) The altds directive allows you to further constrain the cluster identification string for uniqueness. In most instances, the cmsd generates a globally unique cluster identification string. However, depending on the configuration that may not be possible (e.g., two separate clusters using the same meta-manager as their manager). The cittag directive allows you to further differentiate the cluster identification to make sure it is unique across your clusters.

**Example**
cms.cidtag dpm01
### 4.4 conwait

**Function**
Set the number of second to delay an `xrootd` client in the absence of a manager `cmsd`.

**Parameters**

- **sec** The number of seconds that a client is delayed when there is no connection to a manager `cmsd`. The time may be suffixed by `s` (the default), `m`, or `h` to indicate seconds, minutes, and hours, respectively.

**Defaults**

- `cms.conwait 10`

**Notes**

1) When a client attempts to locate a file and no connection exists to a manager `cmsd` process, `xrootd` defers the client for `conwait` seconds. After the time period expires, the client automatically retries the request.

2) The time period chosen for `conwait` should be sufficiently long to establish a connection to a `cmsd`.

**Example**

- `cms.conwait 6`
4.5 delay

```
cms.delay parms

parms:  [delnode sec] [discard num] [drop sec]
        [full {sec | *}] [hold msec] [lookup sec]
        [nostage ns cnt] [overload {sec | *}] [peer sec]
        [qdl sec] [qdn num] [servers num[%]] [service sec]
        [startup sec] [suspend sec]
```

Function

Manage processing latency.

Parameters

delnode sec

The maximum number of seconds that cmisd should wait to delete an in-use node object. If the object is still in use after sec, it abandoned and its memory lost. The default is 15 minutes.

discard num

The maximum number of times a message can be forwarded before it gets discarded.

drop sec

The number of seconds a malfunctioning server is allowed to stay in the configuration before it gets dropped. The delay allows time for a server recover before clients are sent to other functioning servers. Clients are delayed during the recovery window.
full \( \text{sec} \)

The number of seconds to delay a client when no eligible servers have sufficient space to place a file. By default, delays due to insufficient disk space are not allowed and when the condition occurs, the client is given an \texttt{ENOSPC} error condition. You may decide that this is a recoverable condition and are willing to let clients wait until disk space becomes available. Specifying an asterisk uses a dynamically computed optimal value (see the notes).

hold \( \text{msec} \)

The number of milliseconds to optimistically hold a file query request waiting for a server to reply that the file is available. Should a server reply within this window, the client is immediately redirected to that server, subject to the \texttt{qdn} value.

lookup \( \text{sec} \)

The number of seconds to delay a client when trying to determine which servers have the requested file on disk.

nostage \( \text{nscnt} \)

Specifies how many staging servers a client may exclude when reselecting a staging server due to an error. The default is 3.

overload \( \text{sec} \)

The number of seconds to delay a client when all available servers are overloaded. Specifying an asterisk uses a dynamically computed optimal value (see the notes).

peer \( \text{sec} \)

The number of seconds to delay a client when resources are not available in the immediate cluster, peers have been specified but no peers are subscribed.

qdl \( \text{sec} \)

The number of seconds by which a query must complete (i.e. \texttt{query deadline}) with a positive response; after which the file is deemed to not exist. By default, the \texttt{qdl} is set to be the same as the \texttt{lookup} value.
Clustering Configuration

qdn num
The minimum number of servers that must have the file in order to redirect the client within the hold period. The default is 1 which causes an immediate redirection when a server indicates it has the requested file (i.e. the fastest responder wins). Values greater than 64 are set to 64.

servers num[%]
The minimum number of servers that must be subscribed for load balancing to be effective. The number may be suffixed with a percent sign. When specified this way, the number of available servers must be no less that the specified percentage of the maximum number of servers ever subscribed to the cmsd manager since startup. This option effectively determines the server quorum necessary for the cmsd to redirect clients.

service sec
The number of seconds to delay a client when fewer than num servers are subscribed.

startup sec
The number of seconds to delay enabling manager service when initially started. This time period allows for servers to subscribe while client requests are delayed. Clients are delayed “service” seconds during this time.

suspend sec
The number of seconds to delay a client when a selected server is in suspend state.

Defaults

```
cms.delay delnode 15m discard 7 drop 10m full 0 hold 178 lookup 5 nostage 3
cms.delay overload * peer 0 qdl 5 qdn 1 servers 80% service 15 startup 90
cms.delay suspend 30
```

Notes
1) This directive is only used by manager-mode cmsd’s.
2) All time values may be suffixed by s (the default), m, or h to indicate seconds, minutes, and hours, respectively.
3) When specified, the qdl value should be greater than or equal to the lookup value.
4) The overload delay is imposed when all eligible servers have a load greater than the one specified maxload on the sched directive.

5) The full and load options allow you to specify an asterisk to choose the optimal delay value. The optimal value is computed as

\[ \text{ping.ptime} \times \text{ping.pcnt} + 30 \]

The value is optimal because the load balancer will see no change in external conditions until this amount of time has gone by. See the ping directive for additional details.

6) Warning: The 80% default for servers works better as more servers join the configuration since more servers can fail before the system enters a holding pattern. For sites with less than 6 servers, you should specify a fixed number.

7) When the system enters a holding pattern, also known as safe-mode, clients are delayed until the conditions causing the situation are removed. For example, when the number of servers falls below the quorum established by the servers option, safe-mode is entered. The system remains in safe-mode until a quorum is re-established.

8) The nostage option provides a limit on how many times a client may reselect a staging server. The notion is that since all staging servers are the same, an error encountered on one would also occur on any other staging server. The default allows for three tries before the error is considered permanent. This avoids needlessly redirecting clients to other servers.

Example

cms.delay lookup 3 full *
4.5.1 Relationship Between hold & lookup Delay vs. qdl

The left-side graphic illustrates the relationship between the hold and lookup delay and the qdl (i.e. query deadline) value. Initially, a client makes a file-oriented request (e.g. open, stat, etc). If no cached information exists about the file the cmsd sets a query deadline qdl seconds into the future and issues a file existence query to its subscribers. The deadline establishes the time at which if no positive response is received the file is deemed not to exist. It then places the client request in a special internal state for hold milliseconds with the expectation of getting a positive response which would direct the client to the correct server. If no positive response is received within the hold period, the client is asked to wait lookup seconds and try again. The client retries after the delay. If no response regarding the file has yet been received and the query deadline has not passed the client is once again told to wait lookup seconds and retry. The graphic shows that the deadline passes at some point during the third lookup delay. So, when the client retries the third time, the client is immediately told that the file does not exist.

There are several important aspects to understand. First, the qdl value works best if it is an integral multiple of the lookup value. The lookup value should be small enough not to impact overall performance but large enough to minimize retries. The qdl value should be no larger than needed for the particular cluster configuration. The default values work quite well for LAN-based clusters. Some tuning may be required for WAN based clusters, especially if they are federated clusters with no deterministic performance characteristics.

The default hold value is also optimized for LAN clusters and works best if positive response times are rather short. Since no more than about 1000 requests can be placed in hold wait, long hold times become ineffective when even a small fraction of file existence requests produce no positive response. Generally, the special hold state does not provide any benefit for WAN based clusters and should left at the default value.
4.6 fxhold

```plaintext
  cms.fxhold noloc ntime[h|m|s] [htime[h|m|s]] | htime[h|m|s]
```

**Function**
Set the time file existence information is to be cached in memory.

**Parameters**

- `ntime`  The number of seconds file non-existence information may be cached and may be no less than 60 seconds. The time may be suffixed by `h`, `m`, or `s` (the default) to indicate hours, minutes, or seconds, respectively. The default is `htime`.

- `htime`  The number of seconds file existence information may be cached. The time may be suffixed by `h`, `m`, or `s` (the default) to indicate hours, minutes, or seconds, respectively.

**Defaults**
```
cms.fxhold 8h
```

**Notes**

1) This directive is only used by manager-mode `cmsd`’s.
2) The time limit for non-existence starts after the cache entry has been fully validated. A cache entry is considered partially validated when a file search is in progress or when server transitions are occurring.
3) A manager `cmsd` keeps track of where files are at each server-mode site. To prevent information from getting very stale, it is discarded after the time specified by the `fxhold` directive.
4) Setting the cache time too low will substantially increase overhead.

**Example**
```
cms.fxhold 3h
```
4.7 fsseq

```
cms.fsseq \{ \func \} \xpath
```

**Function**
Designate the program to handle file meta-data operations.

**Parameters**

*func* One or more of the indicated functions (i.e., *chmod*, *mkdir*, *mkpath*, *mv*, *rm*, *rmdir*, and *trunc*) that are to be handled by *xpath*.

*xpath* The absolute path to an executable file. The file will be invoked whenever the *cmsd* is asked to execute one of list functions. Parameters specified after *xpath* are passed to the program via the argument list. Quoted parameters should not be specified as these are not correctly passed.

**Defaults**

*None*. The *cmsd* will either use the native operating system call or the local *xrootd* server to perform the functions.

**Notes**

1) This directive is only used by server-mode *cmsd*’s.
2) Any number of *fsseq* directives may be specified in order to map different programs to different functions.
3) The *fsseq* directive is meant to be used in those situations where additional processing needs to occur when one of the indicated functions is executed (e.g., a file needs to be deleted from online disk as well as a Mass Storage System).
4) The *cmsd* is asked to execute functions only if the *ofs.forward* directive has been specified for the redirecting file server (e.g., *xrootd*). Refer to the *ofs* configuration manual for more information.
5) Each function invokes \textit{xpath} as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Command Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmod</td>
<td>\textit{xpath mode} \textit{path}</td>
</tr>
<tr>
<td>mkdir</td>
<td>\textit{xpath mode} \textit{path}</td>
</tr>
<tr>
<td>mkpath</td>
<td>\textit{xpath mode} \textit{path}</td>
</tr>
<tr>
<td>mv</td>
<td>\textit{xpath oldpath_newpath}</td>
</tr>
<tr>
<td>rm</td>
<td>\textit{xpath path}</td>
</tr>
<tr>
<td>rmdir</td>
<td>\textit{xpath path}</td>
</tr>
<tr>
<td>trunc</td>
<td>\textit{xpath size path}</td>
</tr>
</tbody>
</table>

6) The executable function must return a status code of zero upon success. Upon failure, the status code should map to the appropriate \texttt{<errno.h>} code that describes the failure.

Example

\texttt{cms.fsseq mv rm /usr/local/bin/fs\_cmsd \_c /opt/fs/fs.cf}
4.8 namelib (an oss directive)

```
oss.namelib path [parms]
```

**Function**

Specify the location of the file name mapping layer.

**Parameters**

- **path** The absolute path to the shared library that contains an implementation of the Name2Name interface that cmsd is to use to make logical file names to physical name for file system specific operations (e.g., open, close, read, write, rename, etc).

- **parms** Optional parameters to be passed to the Name2Name object creation function.

**Defaults**

A built-in minimal implementation driven via the localroot and remoteroot directives is used.

**Notes**

1) The Name2Name interface is defined in XrdOucName2Name.hh include file. Refer to this file on how to create a custom file name mapping algorithm.

2) The Name2Name interface is also used by the oss component of xrootd.

3) The cmsd honors the oss namelib directive. This allows you to use a single configuration file for the cms and oss components.

**Example**

```
oss.namelib /opt/xrootd/lib/libN2N.so
```
4.9 nbsendq

```
cms.nbsendq {all | off | remote} [maxq {mq | none}] [warn wnum]
```

**Function**
Specify non-blocking send queue parameters.

**Parameters**

- **all**
  Uses non-blocking sends for all LAN and WAN control messages.

- **off**
  Never uses non-blocking sends for control messages.

- **remote**
  Uses non-blocking sends for all WAN control messages; blocking sends are used for LAN control messages. This is the default.

- **mq**
  Is the maximum number of messages that may be queued for sending should the connection be blocked. Any additional messages past this number are discarded. If **none** is specified, messages are never discarded since no limit applies.

- **wnum**
  Issues a warning after this number of messages are queued.

**Defaults**
```
cms.nbsendq remote maxq 30 warn 3
```

**Notes**

1) The **cmsd** sends control messages for various actions such as file location, file preparation, and location cache management, among other actions. During periods of high activity, the number of messages that are sent may exceed the speed or quality of the network connection to a particular server causing sending to block. When sending is blocked, the **cmsd** may suffer a severe slowdown in overall performance. This typically may occur on WAN connections but rarely, if ever, on LAN connections.
2) In order to avoid blocking on a slow connection, the \texttt{cmsd} uses non-blocking sends for WAN connected servers. LAN connected nodes use, by default, blocking sends to avoid message management overhead. The \texttt{nbsendq} directive allows you to change this logic.

3) When a control message is discarded, the \texttt{cmsd} loses any information related to the message. For instance, a file lookup message, when discarded, would make it appear as if the target server does not have the associated file. Hence, it is important to minimize discarding of messages. The default of 30 queued messages should rarely be reached and if it is reached, it typically indicates that the server associated with the connection is unreliable and should likely not be used. So, lost messages are acceptable in this case.

4) In order to avoid flooding the log with warning messages, the \texttt{cmsd} uses a progressive reduction of such warnings. The first message will appear when \texttt{wnum} messages are queued. A subsequent warning will appear when double that number is queued. Subsequent warnings require even more messages to be queued. The counter controlling the warning messages is periodically reset.

Example
\begin{verbatim}
cms.nbsendq remote maxq 60 warn 10
\end{verbatim}
4.10 nowait

Function
Specify that the cmsd should not wait for the data server.

Defaults
None, you must specify the nowait directive or start the cmsd with –i to not wait for a data server.

Notes
1) The nowait directive provides for a loose coupling between servers running on the same host. The cmsd executes asynchronously from the host’s data server and can subscribe to a manager before the data server is available on the host.
2) Without nowait, a host is not available for selection until the host’s data server is ready.
3) Once the xrootd contacts the cmsd, the host automatically becomes ineligible for selection whenever the data server becomes unready.
4) The nowait option is meant for to be used with data servers that are unable to communicate with the local cmsd. You should not specify this option for the xrootd server.
5) Warning: the default cmsd mode (i.e., wait for data server) must be used in conjunction with xrootd’s –t option; otherwise the host will never be selected by the manager cmsd.
6) Warning: The nowait directive disables port remapping. With port remapping, a client is redirected to the port actually being used by the data server that is the target of the redirection. This allows arbitrary or hidden ports to be used, none of which need be the same. When port remapping is disabled, clients are always redirected to the port they initially used to contact the redirector.
7) The nowait directive is automatically implied if you start the cmsd with the –i option.

Example
 cms.nowait
4.11 **osslib** (an *ofs* directive)

```
    ofs.osslib  path  [parms]
```

**Function**

Specify the location of the storage system interface layer.

**Parameters**

- **path** The absolute path to the shared library that contains an implementation of the storage system interface that *ofs* is to use for storage access for file system specific operations (e.g., open, close, read, write, rename, etc).

- **parms** Optional parameters to be passed to the storage system object creation function.

**Defaults**

A full-featured built-in implementation is enabled for use by the *cmsd*.

**Notes**

1) The storage system interface is defined in the *XrdOss.hh* include file. Refer to this file on how to create a custom storage system implementation.

2) A *cmsd* can automatically become a proxy for another manager *cmsd* if the *osslib* implements a proxy mechanism. If you decide to run a proxy *cmsd* then it and its *xrootd* counterpart should be configured with a role of server.

3) The *cmsd* does not support plug-in stacking. Stack specifications are ignored and the *cmsd* only uses the base plug-in.

**Example**

```
    ofs.osslib /opt/xrootd/lib/libmyOss.so
```
4.12 pidpath

```
all.pidpath path
```

**Function**
Specify the location of the pid file.

**Parameters**

- `path` The path to be used to create the file where the daemon’s process id and local prefix are stored.

**Defaults**
The process id file is written into `/tmp`.

**Notes**

1) The name of the pid file is determined by the `cmsd`’s role and the `-n` option.
2) If the `cmsd` cannot create the pid file because either one already exists but is not owned by the `cmsd`, or the directory permissions prohibit the `cmsd` from creating new file; initialization fails and the `cmsd` exits.
3) To create a specific `pidpath` exception for the `cmsd`. Use the “`cms`” prefix instead of “`all`”.

**Example**

```
cms.pidpath /var/run/cmsd
```
### 4.13 ping

**Function**
Control the keep-alive and load reporting frequency.

**Parameters**
- `ptime`  The time between keep-alive requests sent to each server `cmsd`. The time may be suffixed by `s` (the default), `m`, or `h` to indicate seconds, minutes, and hours, respectively.

- `log ucnt`  The number of usage requests that must be made before the reported usage is logged. A value of 0 suppresses any logging of usage information.

- `usage pcnt`  The number of pings that must occur before usage is requested from a server `cmsd`. A value of 0 suppresses usage requests.

**Defaults**
```
cms.ping 60 log 10 usage 10
```

**Notes**
1) This directive is only used by manager-mode `cmsd`'s.
2) Unspecified values in subsequent ping directives default to the last known value.
3) Smaller `ptime` values will discover a failing `cmsd` is a smaller time window at increasing overhead.
4) Smaller `pcnt` values will ask for usage information averaged across a smaller time-window.
5) Usage information will be requested every `pcnt*ptime` seconds, assuming `ptime` is in seconds. Select a `pcnt/ptime` value that averages usage across a reasonable time window for your load (e.g., 5 to 10 minutes).
6) Usage information for each cmsd server will be logged every 
ucnt*pcent*ptime seconds, assuming ptime is in seconds. Choose any value 
appropriate to your logging needs. For instance, 1 logs usage every time it 
is requested while 0, the default, does not log usage.

7) When pcent or ptime is set to zero, usage based load balancing is disabled. 
This means that requests are scheduled round-robin.

8) In the subsequent example, keep-alive pings occur every 30 seconds. 
Usage is requested every five minutes and never logged.

Example

cms.ping 30 log 0 usage 10
4.14 prepmsg

```plaintext
cms.prepmsg msgline

msgline: [text] [var] [msgline]

var:    $CGI | $LFN | $PFN | $RFN | $NOTIFY | $OFLAG | $PRTY | $RID | $eVar
```

Function
Specify the message to be sent to a piping `prep ifpgm` when a prepare request is received.

Parameters

- `text` Arbitrary text.
- `var` A variable whose value is determined by the current request setting. The following variables may be specified:
  - `$CGI` all of the opaque information specified after the question mark in the file path
  - `$LFN` logical file name
  - `$PFN` physical file name as modified by `localroot` or the `namelib` plug-in
  - `$RFN` remote file name as modified by `remoteroot` or the `namelib` plug-in
  - `$NOTIFY` notification string; as follows:
    - no notification is to be sent.
    - file://path send an ofs event message via a Unix pipe named path
    - mailto://user send e-mail to user
    - tcp://rhost:port/msg send msg via tcp to rhost:port
    - udp://rhost:port/msg send msg via udp to rhost:port
  - `$OFLAG` a character sequence describing the file open processing flags:
    - w - O_WRONLY | O_RDWR
    - r - O_RDONLY
  - `$PRTY` request priority
  - `$RID` request identifier
$eVar any variable that has been passed along with the file name as opaque information

Defaults
+ $RID $NOTIFY $PRTY $OFLAG $LFN

Notes
1) Variables must begin with a $ (dollar sign) and end with a non-alpha-numeric character.
2) To include a dollar sign into the message, escape it with a back slash ("\").
3) A backslash escape is only recognized when followed by a dollar sign.
4) Important! The prepmsg msgline is not subject to general set variable substitution.
5) Except for $CGI, the implicit value of a variable that has not been set is the variable name itself, including the dollar sign.
6) For $CGI, if no opaque information is found, the variable is substituted with the null string.
7) The default prepmsg slightly differs from the one given above in that $OFLAG contains additional information. See the description of mode under the prepare directive for additional information.

Example

cms.prepmsg prepare $LFN $PFN $RFN
4.15 remoteroot (an oss directive)

**Function**
Specifies where the local file system name space is actually rooted in the remote Mass Storage System.

**Parameters**

*path* The path to be pre-pended to any path sent to the Mass Storage System for processing.

**Defaults**
None. Paths are sent to the Mass Storage System as specified.

**Notes**
1) The `remoteroot` parameter allows you to place the online file namespace in a different location within the Mass Storage System. Say that the online file system is mounted at `/xrd`. This means that all file paths start `/xrd`. If you specified

```
oss.remoteroot /usr
```

then the file namespace would be rooted at `/usr/xrd` within the Mass Storage system because all paths would be prefixed by `/usr` before being sent to the Mass Storage System for processing.

2) The `cmsd` honors the `oss remoteroot` directive. This allows you to use a single configuration file for the `cms` and `oss` components.

**Example**
```
oss.remoteroot /usr
```
4.16 repstats

```
cms.repstats [-]soption [ [-]soption ] [ · · · ]
```

```
soption: all | frq | shr
```

**Function**

Enable additional statistical reporting.

**Parameters**

`soption`

The additional statistics to be reported when `xrd.report` specifies protocol summary reporting. One or more options may be specified. The specifications are cumulative and processed left to right. Each option may be optionally prefixed by a minus sign to turn off the setting. Valid options are:

- `all` : all possible additional information
- `frq` : information about the fast response queue
- `shr` : share usage

**Defaults**

`cms.repstats -all`.

**Notes**

1) See the `xrd.report` directive in the Xrd/Xrootd reference on how to turn on protocol summary reporting.
2) When protocol summary information is turned on, the `cmsd` reports basic information that is usually sufficient for monitoring purposes. The `repstats` directive allows you to request additional information that may be useful for tuning purposes.
3) The `frq` information is only available for `cmsd`’s with a manager or supervisory role.
4) The `shr` information is only available for meta-manager `cmsd`’s.
5) The Monitoring Reference on more information about the reported statistics.

**Example**

```
cms.repstats shr
```
4.17 request

\begin{verbatim}
cms.request [delay secd] [fwdwait msf] [noresp num]
[prepwait msp] [repwait secr]
\end{verbatim}

Function
Specify request timing parameters.

Parameters

- **secd**: The number of seconds to delay an \texttt{xrootd} client when the \texttt{cmsd} has not responded in \texttt{secr} seconds to a request to locate the file the client wishes to access. The time may be suffixed by \texttt{s} (the default), \texttt{m}, or \texttt{h} to indicate seconds, minutes, and hours, respectively.

- **msf**: The number of milliseconds of wait time to impose between forwarded requests (i.e. mv, rmdir, and rm).

- **num**: The number of consecutive \texttt{secr} \texttt{cmsd} response timeouts that may be tolerated before \texttt{xrootd} attempts to find another working \texttt{cmsd} manager.

- **msp**: The number of milliseconds of wait time to impose between prepare requests.

- **secr**: The maximum number of seconds to wait for a \texttt{cmsd} response. The time may be suffixed by \texttt{s} (the default), \texttt{m}, or \texttt{h} to indicate seconds, minutes, and hours, respectively.

Defaults
\texttt{cms.request delay 5 fwdwait 0 noresp 4 prepwait 33 repwait 3}

Notes
1) When a client attempts to locate a file a request is sent to the \texttt{cmsd} to locate the best possible copy of the file. Should the \texttt{cmsd} not respond in \texttt{secr} seconds, \texttt{xrootd} defers the client for \texttt{secd} seconds. After the time period expires, the client automatically retries the request.

Example
\texttt{cms.request delay 3 repwait 1}
4.18 subcluster

```
all.subcluster [of] host[+]{:port | port]
```

**Function**

Define a subordinate cluster that is actually part of another cluster.

**Parameters**

- **host**  
The DNS name or IP address of the cmsd manager of the cluster that is to accept this subordinate cluster. If `host` ends with a plus sign (+), then the all hosts addresses associated with `host` are considered to be available managers.

- **port**  
The TCP port number or service name at which the manager will accept connections. The port may be specified with an adjacent colon or space separation.

**Defaults**

None; see the Notes for requirements.

**Notes**

1) The `subcluster` directive is processed only for simple `manager` roles (i.e. not qualified in any way); otherwise, it is ignored.

2) A subordinate cluster may only join managers within the same DNS domain. Cross-domain clusters are not allowed.

3) The `subcluster` directive is cumulative in that the specified managers are additive.

4) Subordinate clusters are useful for independently defining a special cluster and then making it part of a larger cluster. For instance, a special cluster could be one whose servers all have the same type of storage device (e.g. SSD) and need to be managed as a unit.

5) This directive must be visible in the cmsd and xrootd configuration files.

**Example**

```
all.subcluster of headmanager.slac.stanford.edu:1213
```
4.19 superport

```
cms.superport port [ if conds ]
```

**Function**
Specify a supervisor’s TCP port number.

**Parameters**
- **port** The TCP port number or service name at which the supervisor will accept connections.
- **conds** The conditions that must exist for this directive to apply. Refer to the description of the `if` directive on how to specify `conds`.

**Defaults**
An arbitrary port is used.

**Notes**
1) The `subcluster` directive is applicable only for `supervisor` roles.
2) Normally, supervisors can use an arbitrary port and this is the common mode of operation. The support directive allows you to specify a specific port should the need arise.

**Example**
```
cms.superport 1717
```
4.20 vnid

```
cms.vnid {=id | <path | @libpath [parms]}
```

**Function**

Specify the unique virtual network identifier for the **cmsd** node.

**Parameters**

- `=id` specifies the actual identifier.
- `<path` specifies the path to a file that contains the identifier.
- `@libpath` specifies the path to a shared library plug-in that supplies the identifier.

- `parms` are optional parameters that are to be passed to the plug-in identified by `libpath`.

**Defaults**

The virtual network identifier is the node’s IP address and host name, if any.

**Notes**

1) Virtual network identifiers may not exceed 64 characters and must be composed of letters, digits, and punctuation characters excluding ampersand (`&`) and the space character.

2) A virtual network identifier needs to be specified if the **cmsd** is running in a virtual machine or a container and its IP address or host name may change when it is restarted or relocated. Refer to the next section for more information.

3) The **vnid** plug-in interface is defined in **XrdCmsVnId.hh** include file.

**Example**

```
cms.vnid =xyzzy/foof.ulumi
```
4.20.1 Using Virtual Network Identifiers

The `cmsd` is responsible for determining the location of each requested file or resource. To speed this function, the manager and supervisor `cmsd`’s maintain a cache of recently looked-up names (i.e. files or resources). The cache cross references each name to the locations providing service to the name. When a server exists the cluster, all resources associated with the server a made invisible for a configurable amount of time (default of 10 minutes) to allow the server to re-enter the cluster and revalidate all cache entries associated with the server. If the server does not re-enter the cluster after the configurable deadline, all cache entries associated with the server are purged. This mechanism provides maximum flexibility and performance.

The `cmsd` tracks cache entries relative to a server by the server’s IP address. This works well for static environments where the IP address and DNS name are predictable. Unfortunately, this is not always the case for servers that execute n virtual machines or containers. For instance, in many environments virtual machines may be relocated to a different physical host with a resultant IP address change. In many containerized environments, IP addresses, as well as host names, are reassigned when a container is restarted. Such occurrences make it impossible to consistently track cache entries using a server’s IP address or even host name.

The `cmsd` solves these kinds of problems by providing a virtual network address space. A virtual network is simply an external namespace overlaid on top of the physical IP address network. This allows IP addresses and DNS names to change as long as the virtual network identifier assigned to the server remains constant. Associating the virtual network identifier with cache entries allows the IP address and DNS name to change without invalidating the cache entries or, worse yet, associating them with the wrong server.

Since control the virtual network identifier resides with the server, not the VM or container infrastructure, a consistent network topology can be maintained. You should assign a unique virtual network identifier to each node in your configuration under the following conditions:

- servers run in virtual machines that can be relocated or do not have fixed IP addresses,
- servers run in containers within an infrastructure that assigns an arbitrary IP address when the container is restarted, or
- servers run in containers with host networking enabled but can be relocated to a different host.
Additional considerations should be given to environments that export resources (e.g. files) through arbitrary servers (e.g. when a file system can be associated with any server at any time). This essentially destroys any association between a server and the resources it is providing. In such cases, the virtual network identifier should be associated with the resource (e.g. file system) not the server that is exporting the resource.

The vnid directive provides various ways to establish the virtual network identifier. The least flexible is to specify the identifier in the server’s configuration file since that ties the server to the resource it is exporting. If you need to associate the virtual network identifier with the resource for maximum flexibility, choose either the file mechanism or write a plug-in to supply the correct virtual network identifier.

Containerized environments pose additional challenges when the cmsd’s companion xrootd is run in the separate container. This potentially allows the wrong xrootd to be associated with a cmsd upon restart of either one. By assigning the same virtual network identifier to the xrootd and its specific cmsd, the system can verify that a consistent network topology is being maintained regardless of how IP addresses of DNS names change. To appreciate this problem, imagine an xrootd that exports file system X connects to a cmsd which reports to its manager that it is managing file system X. Then upon restarting the xrootd, it happens to connect to a cmsd that reported that it is managing file system Y due to a previous xrootd connection that no longer exists. Clearly, any cached information about locations in file system X or Y will be incorrect. Such problems can be mitigated by making sure that each xrootd and its companion cmsd have the same virtual network identifier. Of course, such problems cease to exist when the xrootd and its companion cmsd reside in the same container; which is the recommended setup.

Finally, you must make sure that virtual network identifiers are unique. The system rejects duplicate network identifiers much like TCP rejects duplicate IP addresses. However, just like TCP it is impossible to fully reject messages sent by nodes with duplicate virtual network identifiers. This may lead, as expected, to undefined behavior.
4.20.2 Virtual Network Identifiers and Kubernetes

The Kubernetes container deployment framework allows you to easily specify virtual network identifiers via the label attribute you can assign to pods. Each pod is assumed to contain all the components that are required for a functioning XRootD instance. It also assumes that the exported resource (e.g. file system) is tightly bound to the XRootD server in the defined pod. If all of this is true then you can use a pod label in its yaml configuration file to assign the virtual network identifier. This is known as a downward API specification. So, in a typical yaml file defining a server container you would typically specify the following (note that metadata.name should be unique to each pod):

```yaml
apiVersion: v1
derived: Pod
description:
  name: XRootD.0001
description:
  containers:
    - name: test-container
      image: XrootD-image
      command: ["/bin/sh", "-c", "env"]
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name

  ...
```

Then in your XRootD configuration file you can specify

```plaintext
cms.vnid =${MY_POD_NAME}
```

This works under the following assumptions:

a) Each pod represent a unique resource provider (i.e. XRootD plus cmsd instance),

b) The resource is tightly bound to the containers in the pod (i.e. file system).

c) Each pod name is absolutely unique.
4.21 trace

cms.trace [-]toption [ [-]toption ] [ • • • ]
tooption: all | debug | defer | files | forward |
    redirect | space | stage

Function
Enable tracing.

Parameters
tooption
The tracing level. One or more options may be specified. The specifications are cumulative and processed left to right. Each option may be optionally prefixed by a minus sign to turn off the setting. Valid options are:
all selects all possible trace levels except debug
debug traces internal functions in cmsd and the xrootd cmsd client
defer traces imposed wait responses in cmsd
files traces file location requests and responses
forward traces forwarded functions in the xrotd cmsd client
redirect traces request redirection in the xrootd cmsd client
space traces changes in space utilization in the xrootd cmsd client
stage traces binding of locate requests to servers to have promised to stage in files in cmsd.

Defaults
cms.trace -all.

Notes
1) The cmsd –d command line option is equivalent to cms.trace all debug.

Example
cms.trace debug
4.22 whitelist

```plaintext
cms.whitelist [check sec] [path]
```

**Function**
White list one or more nodes.

**Parameters**
- `sec` is the amount of time between checks whether or not the whitelist file has been changed. When a change is detected, the file is reprocessed and the whitelist updated. The time may be suffixed by `s` (the default), `m`, or `h` to indicate seconds, minutes, and hours, respectively. The default is 10 minutes (i.e. 10m) and may not be less than one minute.

- `path` is the absolute path of the whitelist file. The default is name of the whitelist file is “cms.whitelist” which is assumed to exist in the configuration file directory.

**Defaults**
```
cms.whitelist check 10m configdirpath/cms.whitelist
```

**Notes**
1) White-listing is not applied unless the `cms.whitelist` directive is specified. You need not specify any options if the defaults are acceptable.

2) If the configuration file contains a `cms.blacklist` directive as well as a `cms.whitelist` directive, the last such directive applies.

3) Refer to the following section on how to code a whitelist file.

4) The `cms.whitelist` directive only applies to nodes with a manager or meta-manager role.

5) White-listed nodes are allowed to login. Nodes that do not match any specification in the whitelist are prohibited from logging in (i.e. they are blacklisted).

6) White-listed entries may be redirected to another cluster. If this occurs, no login retries are attempted at the redirecting host.

7) Redirection is only supported for CMS clients at version 4.2 or above. Clients below this version are effectively blacklisted and not redirected.
8) Nodes that are already logged in and found to no longer be white-listed and not redirected are disabled and forced to logoff.
9) Nodes that are already logged in and found to be redirected are asked to disconnect and retry the login; which causes a redirect. If the node does not disconnect within the ping interval, it is forcibly disconnected.
10) If the whitelist file is not present, no controls are applied (i.e. all connections are allowed to login).
11) If the whitelist file is present but contains a syntax error or cannot be read, the current white-list is not changed.

Example

cms.whitelist /var/run/cms.whitelist
5 Blacklist and Whitelist File Format

The black or white list file consists of new line separated records. A line may be blank, contain a comment (i.e. the first non-space character is a pound sign, #), or contain a single specification of the host(s) that are blacklisted or whitelisted. The format of the specification is shown below.

```
hostspec | [hpfx]*[hsfx] [redirect target]
hostspec: fulldnsname | [ipv6address] | ipv4address
target: fulldnsname[+]:port [target]
```

Parameters

**hostspec**

The DNS registered name or IP address. IPV6 addresses must be surrounded by brackets. The host matching this specification is either blacklisted or whitelisted. In general, blacklisting and whitelisting work best when DNS names are used.

**hpfx**

The starting characters of a DNS registered name. If specified, the leading characters must match the host’s name in order for the record to apply.

**hsfx**

The trailing characters of a DNS registered name. If specified, the trailing characters must match the host’s name in order for the record to apply.

**target**

Where a matching host is to be redirected. If a manager node target is replicated, you should specify all of the replicated nodes either by listing individually them or using a DNS alias that associated all of the replicas together. If you are using a DNS alias, you must specify a plus sign (+) after the DNS alias name.

Notes

1) The **cms.blacklist** directive prohibits hosts with a matching entry in the file from logging in. The **cms.whitelist** directive does the opposite; hosts with a matching entry in the file are allowed to login.

2) Host matching occurs in the same order as the entries appear in the file.
3) When you redirect to multiple targets, the host assumes that these are replicas and function identically. It disconnects from all managers associated with the manager that provided the redirect, then connects to each of the specified nodes, and resumes normal operation.

4) Redirecting to functionally non-identical nodes will produce non-deterministic file look-up behavior and should not be done.

5) redirection works for black lists or white lists.

6) You may not specify more than 255 lexically different redirect targets.

7) Redirection is only supported for CMS clients at version 4.2 or above. Clients below this version are only blacklisted and not redirected.

8) To safely update the blacklist or whitelist file in-place, follow the steps:
   a. Copy the existing file to a temporary location in the same directory,
   b. Update the copy as needed, and
   c. Rename the temporary copy to the original name using the mv command or rename() function.

Example

```plaintext
# Apply rule to a single host
#
foobar.slac.edu

# Apply rule to a domain
#
*infn.it

# Apply rule to a group of hosts
#
worker*.slac.edu

# Redirect a domain to a replicated manager
#
*google.com redirect manager.cern.ch+:1213

# Specifically for the white list, allow a domain but
# redirect all other hosts elsewhere via two entries
#
*slac.edu
  * redirect manager.bnl.gov:1213
```
6 Document Change History

26 October 2007
- New manual to document the Cluster Management Service.

1 December 2007
- Describe the meta manager attribute in the manager and role directives.

8 December 2007
- Describe the seclib directive.

8 January 2008
- Add documentation on StartCMS and StopCMS.
- Document the prepmsg directive.
- General cleanup.

7 April 2008
- Document trunc under the fsseq directive.

11 April 2008
- Document min% and hwm% in the space directive.

6 January 2009
- Correct description of the default prepmsg.
- Change priority scale from 0-9 to 0-2.
- Deprecate mps_prep and mps_PreStage as frm_psgta and frm_pstgd have replaced them.

21 April 2009
- Document XrdCnsd and the cnsssi command.

5 October 2009
- Document the –L, -N, and –R options of the XrdCnsd command.

17 November 2009
- Document the –B option of the XrdCnsd command.
17 March 2010
• Preferentially document ‘all.pidpath’ as opposed to ‘cms.pidpath’.
• Minor text corrections.

26 April 2010
• Document oss.space directive instead of deprecated oss.cache directive.
• Document the new built-in prepare mechanism based on the File Residency Manager.
• Minor text corrections.

6 January 2011
• Document the ‘ofs.osslib’ directive for cmsd use.

22 February 2011
• Document the ‘cms.dfs’ directive for cmsd use.

8 March 2011
• Document the ‘cms.delay qdl’ option.
• Document the –b, –p, and –s command line options.
• Document the fwdwait option of the cms.request directive.

6 June 2011
• Document the ‘cms.delay qdn’ option.
• Document the ‘cms.shed gsdlft’ option.
• Document the ‘cms.shed gshr’ option.
• Document the ‘cms.repstats’ directive.

6 June 2011
• Correct example of using the XrdCnsd as a command.

----------------
Release 3.1.0
----------------
Release 3.1.1
----------------
Release 3.2.0
----------------
Release 3.2.1
----------------
Release 3.2.2
----------------
Release 3.2.3
----------------
Release 3.2.4

23 September 2012
• Remove unneeded directive in configuration examples.
Clustering Configuration

Documentation Changes

-------------  Release 3.2.5
-------------  Release 3.2.6
-------------  Release 3.2.7

31 October 2012

- Document the `altds` directive.
- Deprecate the `xmilib` directive.

-------------  Release 3.3.0
-------------  Release 3.3.1
-------------  Release 3.3.2
-------------  Release 3.3.3
-------------  Release 3.3.4
-------------  Release 3.3.5
-------------  Release 3.3.6

23 February 2013 (IPV6 Introduction)

- Document the `–I` command line option.
- Document the `cache` option in the `xrd.network` directive.

12 August 2013

- Document the extended `–k`, `–I` and `–z` command line options.
- Document exported environment variables.
- Document the environment information file contents.
- General clean-up and better explanations.

6 September 2013

- Simplify the `role` directive by removing the `peer` option.
- Fully explain the `peer` option in the `manager` directive.
- Redefine the `peer` option of the `delay` directive.
- Remove the unsupported `xmilib` directive.

19 January 2014

- Document the `cms.blacklist` directive.

2 April 2014

- Better explain the `manager all` and `any` options.
- Better explain the `delay hold`, `lookup`, and `qdl` options.
Documentation Changes

---

**Release 4.0.0**

**Release 4.0.1**

**Release 4.0.2**

**Release 4.0.3**

**Release 4.0.4**

13 October 2014
- Document the new *subcluster* directive.

15 October 2014
- Change *cms.subcluster* to *all.subcluster* directive so that the OFS and CMS components see the configuration.

---

**Release 4.0.0 to 4.2.3**

9 December 2014
- Document how to support disjoint clusters by using the host name qualifies on the *all.manager* directive.
- Document the *cms.whitelist* directive.
- Document the *redirect* option in the blacklist and whitelist files.

17 November 2015
- Document *cms.cidtag* directive.
- Document the *delnode* option of the *cms.delay* directive.
- Document the *files* option of the *cms.trace* directive.
- Correct the definition of the –d option in the *cms.trace* section.

25 November 2015
- Document nostage option of the *cms.delay* directive.
- Document retries option of the *cms.dfs* directive.
- Explain the side-effects of the –s command line option on the placement of the environmental file.

---

**Release 4.3.0**

10 February 2016
- Document the *mode* option of the *cms.sched* directive.
18 April 2016
- Document log file plug-ins.
- Add admonition of when the `all.role` directive should not be used.

20 June 2016
- Document the `cse` logging plug-in parameter.

-------- Release 4.4.0 --------

21 October 2016
- Document the `noloc` option of the `cms.fxhold` directive.

-------- Release 4.5.0 --------

6 February 2018
- Document the `cms.nbsendq` directive.
- Document the `cms.superport` directive.
- Document the `cms.vnid` directive.
- Add section explaining virtual network identifies.

8 May 2019
- Document the `maxretries` and `nomultisrc` options in the `cms.sched` directive.

21 June 2019
- Add information about running clusters using a container orchestration system and how to stop daemons to the FAQ.

17 September 2019
- Document the `xrootd` and `lib` arguments in the `cms.perf` directive.
- Add missing protocol token, `xroot`, to the `cms.altds` directives.

10 April 2020
- Document the `space` argument in the `cms.trace` directive.

14 April 2020
- Document the `-a`, `-A`, `-w`, and `-W` command line options.
- Document the `xrd.homepath` directive.
14 October 2020
- Document the \%iname specification on the all.manager directive.

14 June 2021
- Document that the mdhold option of the cms.dfs directive is incompatible with http[s].

27 July 2021
- Document that the affpath option of the cms.sched directive.