XRootD

XRootD Configuration Reference

27-October-2017
Release 4.8.0 and above

Andrew Hanushevsky

SLAC
NATIONAL ACCELERATOR LABORATORY

1 Introduction

This document describes the eXtended Request Daemon (**xrd)** configuration directives and the configuration options for two protocols that can be used with **xrd**: **XRootd** and **rootd**.

The **xrd** is a server that can dynamically support multiple TCP/IP application service layer protocols. It is designed to provide a high performance environment for application services. The **xrd** is a generalized daemon and it makes its primary decision on which protocol to support based on the name given to the executable. Currently, the following executable names are fully supported:

- **xrootd** for eXtended Root Daemon and related protocols.

Configuration directives for **xrd** come from a configuration file. The characters "**xrd**" must prefix each directive in the configuration file. This makes **xrd** directives compatible with many servers providing other support services. For example:

Component	Purpose
acc	**Ac**cess control (i.e., authorization)
cms	**C**luster **M**anagement **S**ervices
ofs	**O**pen **F**ile **S**ystem
oss	**O**pen **s**torage **s**ystem (i.e., file system implementation)
sec	**Sec**urity authentication
xrd	E**x**tended **R**equest **D**aemon
xrootd	The **xrootd** protocol implementation.
http	The HTTP protocol implementation.
all	Applies the directive to all of the above components.

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**, **http**, **xrd**, and **xrootd** configuration directives (i.e., the un-shaded rows). Other directives are documented in supplemental guide specific to the component they deal with.

The location of the configuration file is specified on the **xrootd** command line. Refer to the reference manuals for other components on how they locate their respective configuration files. Because each component has a unique prefix, a common configuration file can be used for the whole system.

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.

1.1 Security Considerations

Xrd relies on the loaded protocol(s) for strong authentication (e.g., Kerberos, GSI, etc.). Therefore, security is a protocol issue. The **xrootd** and **rootd** protocols provide strong authentication should you choose to use it. Refer to each protocol on how to configure strong authentication.

Xrd does provide host-based authentication. While this type of authentication can be subverted in a number of ways, it still is a practical mechanism for installations that do not need strong authentication. The **allow** directive can be used to restrict the range of hosts that can connect to the daemon. This security can be used together with any protocol-provided security.

Because **xrd** does not intrinsically provide strong authentication; you *should not run* **xrootd** *as super-user* (i.e., Unix root). Any attempt to do so without indicating that you *really* want to run super-user (see the **-R** command line option) will cause the program to exit.

1.2 Starting the xrootd Daemon

Use the following command to start the **xrd**-based **xrootd** daemon:

```
xrootd [ options ] [ path [ path [ . . . ] ] [piargs]

options:   [-c cfn] [-l largs] [-k {num | sz{k|m|g} | sig}]
           [esoteric]

esoteric:  [-b] [-d] [-h] [-I {v4 | v6}] [-L protlib]
           [-n name] [-[-p {port | any}] [-P protocol]
           [-R user] [-s pfn] [-S site] [-z]

largs:     [=]fn | - |
           @lib[,bsz=sz][,cse={0|1|2}][,logfn=[=]fn]

sig:       fifo|hup|rtmin|rtmin+1|rtmin+2|ttou|winch|xfsz

piargs:    -+[tag] [args] [piargs]
```

Parameters

path An absolute file system path prefix. All requests will be restricted to files with this prefix. You may specify any number of path prefixes. If no path is specified, operations will be restricted to paths starting with **/tmp**.

Options

-c *fn* The name of the configuration file. If one is not specified, no configuration file is processed.

-l [=] *fn*

Specified 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). If *fn* is a dash (-), output is sent to standard error; the default.

=*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 **k**ilobytes or **m**egabyte, 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 **k**ilobytes, **m**egabyte, or **g**igbytes, 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

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

-d Turns on debugging.

-h Displays help information.

-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.

-L *protlib*

Specifies the shared library that holds the implementation of the default protocol specified by the **–P** option.

-n *name*

Assigns *name* to the **xrootd** instance. By default, the **xrootd** instance is unnamed. See the notes on how to use this option.

-p *port*

The TCP port, or service name associated with a port, that **xrootd** is use for new connections. The default is "**xrootd**" or port **1094**, if the TCP service **xrootd** cannot be found /etc/services.

-p any
Uses any available port.

-P *protocol*
The name of the default protocol. Use this option when the name of the executable differs from the name of the default protocol. You may need to specify the **–L** option as well. See the notes for more information.

-R *user*
The user name or numeric uid of the *user* whose effective identity is to be assumed. See the usage notes for more information. The specified user may not have super-user privileges. This option may *not* be specified unless the program is running as super-user.

-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.

-z provides microsecond resolution for log file message timestamps.

-+ provides a mechanism to pass command line arguments to plug-ins. See the section "Passing Plug-In Command Line Arguments" for more information.

Defaults

```
xrootd –l - –I v6 –p xrootd –P xrootd /tmp
```

General Notes

1) For security purposes, only files in **/tmp** are allowed to be accessed unless you specify otherwise. You may specify other paths either on the command line or using the **xrootd export** configuration directive.
2) Do ***not*** prefix any export *path* with the **oss localroot** directive path, if any.
3) If a log file is specified without a signal **-k** option, the file is closed at midnight, renamed to have a date suffix (i.e., *fn***.***yyyymmdd*) and possible sequence number (i.e. *fn***.***yyyymmdd.n*), and a new log file is opened.

4) 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* plugin. 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.
5) 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.
6) The *sig* names, except for **fifo**, be fully capitalized as well prefixed by "**sig**" or "**SIG**" when capitalized.

Notes on Esoteric Options

1) The default port service name, default protocol, and *pidfile* name is normally determined by the prefix-name of the executable. The prefix-name is defined to be all of the characters in the base filename (i.e., the directory path removed) up to but not including the first dot in the name, if any. If the name starts with a dot, the prefix-name is the complete base filename.
2) The way the prefix-name is derived allows you to maintain several versions of a particular **xrd** executable (e.g., **xrootd** and **xrootd.debug**) without changing the intrinsic way default names (e.g, protocol) are determined.
3) You must use the **–P** option to set the default protocol name as well as other related naming aspects (e.g., port service name) when the name of the executable does not correspond to the name of the default protocol.
4) The **–n** option allows you to run multiple instances of the **xrootd** on the same machine. See the next section on instances and fencing.
5) Use **-p any** for protocols that manage their own port numbers. This is the case for redirection target **xrootd**/**cmsd** combinations. Only the initial point of contact needs a well-known port number. All other connections between clients and servers are routed using whatever port numbers are currently in effect. This allows you to keep a simple configuration file for servers and to run more than one server on the same machine without worrying about conflicting port numbers.
6) The **–b** option forces the program into the background. If **–l** is not specified; all output messages are discarded.

7) The **-R** option allows the program to run under the super user's account. This is allowed because the effective user is set to specified user and the effective group to user's primary group. Thus, the program is not *effectively* running as super-user. However, the real and saved user ids may still be "root", depending on how the program was started.
8) The **-R** option provides a minimal increase in security since it is possible for a loaded protocol to switch back to super user mode. You should not use the **-R** option unless absolutely necessary.
9) The **–b**, **–p**, and **–s** command line options are meant to be used by start-up scripts (e.g. **init.d**).
10) ***Warning***: Command line options, except for **–s**, over-ride corresponding configuration file directives.

Example

```
xrootd –c /opt/xrootd/xrootd.cf
```

1.2.1 Multiple Instances and Automatic Fencing

You can run multiple instances of **xrootd** on the same physical machine. This is useful when you want to overlay more than one cluster on top of a file system (e.g. production and test). In order to prevent instances from interfering with each other, you must provide each **xrootd** that is running on the same hardware a unique instance name. One of the daemons need not have an instance name as it assumes the name "anon" (i.e. anonymous).

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 threse 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/xrdlog**" is specified along with "**-n test**", **xrootd** modifies the **–l** argument to be **/var/adm/xrootd/test/xrdlog**.

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.

1.2.2 Passing Plug-In Command Line Arguments

You can pass command line arguments to various plug-ins that support the feature using the **-+** option. The option must be specified after all **XRootD** options and parameters as plug-in arguments are stripped from the command line. For example,

xrootd [*options*] [*path* [*path* [. . .]] **-+mypi** *arg1 arg2* **-+urpi** *arg3*

places a pointer to the vector containing "argv[0] *arg1 arg2* 0" into the internal environment passed to the plug-in using the variable "**mypi.argv****" and the count in "**mypi.argc**" (i.e. 3). Similarly, "argv[0] *arg3* 0" are pointed to by "**urpi.argv****" with the count in "**urpi.argc**" (i.e. 2). If no tag is specified, the leading prefix is missing (i.e. the variables are ".**argv****" and "**.argc**"). Note that argv[0] is the actual value of argv[0] (i.e. the executable name) passed to **XRootD**.

It is up to the plug-in to extract the appropriate arguments using a documented tag. Tag values that start with "**xrd**" should not be used as these are reserved for plug-ins normally distributed with the **XRootD** package.

1.2.3 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 "**yymmdd 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.

1.2.4 Files created by xrootd

The following directories and files are created by **xrootd**:

Default File	Changed by	Contents
<*stderr*>	**-l** and **-n** command line options	Informational and error messages
/tmp//[*name*/].xrootd/	**-n** command line option and the **adminpath** directive	Directory for various server-related files.
<cwd>//[*name*/]core[.pid]	**-n** command line options	Core file
/tmp/[*name*/]xrootd.pid	**pidpath** and **–n** option	Holds the process id
/tmp/xrootd.*name*.env	**pidpath** and **–n** option or **–s** directory	Holds environmental information (see next section).

1.2.4.1 Environmental Information File

The daemon writes environmental information in the directory infered 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 "**xrootd.***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.

pid=*pid***&host=***host***&inst=***inst***&ver=***ver***&cfgfn=***cfgfn***&cwd=***cwd***&logfn=***logfn*

Parameters

cfgfn The configuration file used.
cwd The current working directory.
host The host name.
inst The instance name.
logfn The log file being used.
pid The process id.
ver The version string.

1.2.5 Exported Environment Variables

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

XRD Variable	Contents
XRDADMINPATH	Is the directory for log files. By default, it is **XRDBASE/logs**.
XRDCONFIGFN	The effective administrative path used for server management files.
XRDDEBUG	Set to one when the **–d** command line option is specified.
XRDHOST	The current host's DNS name.
XRDINSTANCE	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.
XRDLOGDIR	Is the directory where log files are written.
XRDNAME	The name specified via **–n** or **anon** if no instance name was specified.
XRDPROG	The executable's name.
XRDSITE	The site name specified either via the –s command line option or the **all.sitename** directive.

If the standard **cms** client plugin is being used, the following additional environment variables are exported.

CMS Variable	Contents
XRDCMSMAN	The space separated list of managers for this host each in the form of "*host***:***port*".
XRDCMSCLUSTERID	The globally unique cluster identification for this host.

If the standard **ofs** plugin is being used, the following additional environment variables are exported.

OFS Variable	Contents
XRDROLE	The effective value specified on the **all.role** directive.
XRDTPC	Is set when **TPC** (Third Party Copy) has been configured and represents the version number of the protocol. If the first character is a plus sign, authentication is required.

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

OSS Variable	Contents
XRDN2NLIB	The path and name of the name-to plugin, if specified via the **oss.namelib** directive.
XRDRMTROOT	The local root path specified by the oss.remoteroot directive.
XRDLCLROOT	The local root path specified by the oss.localroot directive.
XRDOSSQUOTAFILE	The name and location of the file handling disk space quotas.
XRDOSSUSAGEFILE	The name and location of the file handling disk space usage.

If the standard **xrootd** protocol plugin is being used, the following additional environment variables are exported.

XROOTD Variable	Contents
XRDOFSLIB	The path and name of the OFS plugin specified by the **xrootd.fslib** directive.
XRDMONRDR	If monitoring is enabled, how often server identification records are sent.

2 Common xrd Configuration Directives

2.1 adminpath

```
all.adminpath path [ group ]
```

Function

Specify the location of the administrative communications path.

Parameters

path The absolute path to a directory that is to hold the Unix named socket used to communicate administrative commands to **xrd** and it related components (e.g., **xrootd**).

group

Allows any user in **xrd**'s group to use a socket in *path* by setting r/w group access on the path

Default (see *warning in the notes*)

/tmp

Notes

1) The **adminpath** directive allows you to specify the location of the local TCP socket used for the command-line administrative functions.
2) ***Warning***: if idle **/tmp** directories and socket files are automatically deleted by the system, you should ***neither*** accept the default *path* ***nor*** allow the **adminpath** *path* to reside in **/tmp**.
3) Local TCP socket names are limited to 108 characters. Up to 32 characters are needed to define actual socket files; leaving 76 characters that may be specified as the *path*.
4) The following steps are taken when creating a Unix named socket in path:
   - If –n is specified, a subdirectory corresponding to the instance name (i.e., **-n** argument) is created in *path*, if one does not exist. This becomes the new *path*.
   - Subdirectories "**.xrd**" and "**.xrootd**" are created in *path*, if they do not exist.
   - Mode bits for these directories are set to 0700 (rwx for owner).

- If group was specified, the mode setting is extended to 0770 (rwx for owner and group).
- A Unix named socket with the name "**admin**" is created in the "**.xrootd**" subdirectory.

5) If the "**.xrd**" or "**.xrootd**" directories already exist, the mode settings are reset to correspond the to the **adminpath** directive.
6) The **adminpath** value is passed to all protocols so that they can create their respective administrative files.
7) Refer to the section "Administrative Interface" for details on how to use the Unix named socket created by this directive.

Example

```
xrd.adminpath /var/adm/xrd group
```

2.1.1 Administrative Interface

The **adminpath** directive is used to construct the path where local TCP sockets, called named sockets, are created. These sockets are used to communicate requests and receive responses via the administrative interface. Unix domain sockets function identically as INET domain sockets. The only differences are in how the socket is created and the domain in which is operates. Care should also be given as to who creates the socket. Following the next steps will allow the successful use of the administrative interface.

1. Wait until the server creates that the socket file (e.g., "/tmp/.xrootd/admin"). This can be done by polling for the socket using **stat()**.
2. Create a stream socket using **socket(PF_UNIX, SOCK_STREAM, 0**)
3. Properly fill out the **sockaddr_un** structure with the path name of the socket (e.g., "/tmp/.xrootd/admin"). This structure is normally defined in the **<sys/un.h>** include file.
4. Issue a **connect()** call to connect the newly created socket to the path.
5. Use **write()** to issue requests to the server and **read()** to read responses.

The Xrootd Protocol Reference defines the administrative protocol used for the socket interface.

2.2 allow

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

Function

Restrict the hosts that can connect to **xrootd**.

Parameters

host *name*

The DNS host name allowed to connect to **xrootd**. Substitute for *name* a host name or IP address. A 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:x]").

netgroup *name*

The NIS netgroup allowed to connect to **xrootd**. Substitute for *name* a valid NIS netgroup. Only hosts that are members of the specified netgroup are allowed to connect to **xrootd**.

Defaults

None. If **allow** is not specified, any host is allowed to connect.

Notes

1) 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 connect to **xrootd**.
2) **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.

Example

```
xrd.allow host objyana*.slac.stanford.edu
```

2.3 port

```
xrd.port [wan] {pnum | any} [ if conds ]
```

Function

Designate the port number to use for incoming requests.

Parameters

wan Sets the preferred **w**ide **a**rea **n**etwork port number. Otherwise, the default port number is set. See the usage notes on how the default is determined.

pnum The **TCP** port number or the **TCP** service name associated with a port in /etc/services file that the daemon should use for incoming requests. See the usage notes on how the default is determined.

any Specifies that any available **TCP** port number may be used use for incoming requests. See the usage notes on how the default is determined.

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

See the usage notes.

Notes

1) The default port number is determined using the following rules:
   - The protocol specified port when the protocol is loaded.
   - The port specified on the protocol directive entry.
   - The port specified on the command line using **–p**.
   - Any available port number if **-p any** was specified on the command line.
   - The port associated with the service name that corresponds to the name of the program used to start the daemon (e.g., **xrootd**).
   - The port value of 1094.

2) The **wan** port number need only be specified if the **xrootd** is behind a firewall and you wish to provide external wide area access at a well known port number. If not specified, the **wan** port is arbitrary and clients discover the port number during the login phase.
3) When running clustered systems, you can keep a single configuration file that is applicable to all types of servers, as follows:
   - always specify "**xrd.port any**" in the configuration file, and
   - use the **if** modifier to identify the top-most servers (i.e., the initial point of contact also known as redirectors) and assign them fixed port numbers immediately following the "**xrd.port any**" directive.
4) Using the steps outlined above, the initial point of contact will have a well-known port number. While all other servers will choose random port numbers, the ports are communicated to the cluster manager which then automatically manages the port numbers while redirecting clients.

Example

```
xrd.port xrdnew
```

3 Esoteric xrd Configuration Directives

3.1 buffers

```
xrd.buffers memsz[k | m | g] [rint[m | s | h]]
```

Function

Limits the amount of memory to be used to data buffers.

Parameters

memsz

The maximum number of bytes to be used for data buffers. The *memsz* can be suffixed by **k**, **m**, or **g** to indicate **k**ilo-, **m**ega-, or **g**iga-bytes; respectively. The default is to use up to 12.5% (one-eight) of the configured memory of the machine.

rint The interval between buffer pool readjustments. Specify a number, optionally suffixed by **m** for **m**inutes, **s** for **s**econds (the default), or **h** for **h**ours. The default is every 20 minutes.

Defaults

xrd.buffers *memsz* **20m**

Notes

1) The allotted memory for buffers is independent of any other memory allotment to the daemon.
2) Data buffers in the pool are periodically readjusted to reflect the actual working needs of the daemon. The *rint* interval controls how frequently this adjustment occurs. The default value is usually the best value.

Example

xrd.buffers 512M

3.2 network

```
xrd.network  [buffsz blen[k | m | g]] [cache sec]
             [[no]dnr] [kaparms idle[,itvl[,cnt]]
             [[no]keepalive]
             [routes {split|common|local} [use if1[,if2]]]
             [[no]rpipa] [wan]
```

Function

Specify network parameters.

Parameters

buffsz *blen*

The buffer size to be set for each connected socket. The *blen* can be suffixed by **k**, **m**, or **g** to indicate **k**ilo-, **m**ega-, or **g**iga-bytes; respectively. The default is determined by the operating system.

cache *sec*

The maximum number of seconds that an address to hostname translation can be locally cached for future use. The *sec* can be suffixed by **s**, **m**, **h** or **d** to indicate **s**econds, **m**inutes, or **h**ours, or **d**ays; respectively. The default is 3 hours.

dnr Uses Domain Name Resolution to convert IP addresses to host name for connecting clients. Host names are displayed in various messages.

nodnr Avoids using Domain Name Resolution to convert IP addresses to host name for connecting clients. Client IP addresses are displayed in various messages.

kaparms *idle*[**,***itvl*[**,***cnt*]

Specifies **TCP keepalive** parameters. The **kaparms** option is only effective for **Linux**. Up to three parameters may be specified. Omitted parameters, as well as parameter values of zero, use the system default. The parameters are:

idle The time the connection needs to remain idle before **TCP** starts sending **keepalive** probes. The *idle* value may be optionally suffixed by **m** for **m**inutes, **s** for **s**econds (the default), or **h** for **h**ours.

itvl The time between individual **keepalive** probes. The *itvl* value may be optionally suffixed by **m** for **m**inutes, **s** for **s**econds (the default), or **h** for **h**ours.

cnt The maximum number of **keepalive** probes **TCP** should send before dropping the connection.

keepalive

Uses the operating system's keep-alive mechanism to determine whether or not a client is still connected to the daemon. This is the default. See the usage notes for other ways of simulating **keepalive**.

nokeepalive

Does not use the operating system's keep-alive mechanism to determine whether or not a client is still connected to the daemon.

routes

Specifies that a dual network exists and how public and private addresses are routed within a site. Select one of the below optionally followed by the **use** keyword and up to two interface names (i.e. *if1* and optionally *if2* immediately preceded by a comma; use **ifconfig** to display interfaces and their names):

split Two separate networks exist. Clients connecting with a private address can only be redirected to a server's private address. Clients connecting with a public address can only be redirected to a server's public address. If clients can use both types of addresses, *all* servers must be dual homed with a public *and* private address. Typically, you must specify the interfaces you are using.

common Two common networks exist. Clients connecting with a private address are preferentially redirected to a server's private address but may be redirected to a server's public address if need be. However, clients connecting with a public address can only be redirected to a server's public address. All servers must have at

least a public address. If the machine is dual-homed you must specify the interfaces you are using.

local Two cross-routable networks exist. Clients connecting with a private address are preferentially redirected to a server's private address but may be redirected to a server's public address if need be. Clients connecting with a public address are preferentially redirected to a server's public address but may be redirected to a server's private address if need be. This is the default mode of operation in order to be compatible with previous releases. If the machine is dual-homed it is advisable to specify the interfaces you are using for predictable access.
Warning, such a network configuration is not suitable for external access and a proxy server must be used for out of domain clients.

rpipa

Tries to resolve private IP addresses to host names. See the section on resolving private IP address for more information.

norpipa

Does not resolve private IP addresses to host names. This is the default.

wan Indicates that the subsequent specifications apply to the special **W**ide **A**rea **N**etwork port. This option is currently not supported by newer clients.

Defaults

```
xrd.network cache 3h dnr norpipa
```

Notes

1) For systems that support TCP buffer auto-tuning as a *manual* option, specify a **buffsz** *blen* of 0 to turn on auto-tuning.
2) Setting the buffer size to a large value may cause the operating system's default value to be used. You should determine the maximum valid value for your system before specifying values greater that 64k.
3) Normally, the best performance is obtained by using TCP buffer auto-tuning.
4) Even if you specific **nodnr**, domain Resolution may be forced on if you specify an **allow** directive using a host name or host name fragment or authorize file access via host names, netgroups, or domain names. Resolution is still optimized by caching the results for future use.

5) The daemon's internal timeout mechanism can be used to discover unconnected clients instead of TCP **keepalive** and may be more responsive. See the **timeout** directive. You should avoid using both mechanisms at the same time.
6) For aggressive **keepalive** processing you can use "**kaparms 300,10,6**".
7) The **wan** port is created only if there is a protocol that can handle wan connections. See the **protocol** directive.
8) By default, public-private networking support is disabled. You must specify the routes option to enable it. If you do specify for one server you must specify for all nodes in your cluster (i.e. servers and redirectors). Failure to do so may result in unreachable nodes..
9) If the supplied information is inconsistent with the server network settings, warning messages are printed and the public-private network support may be turned off for that server. You should verify that your specification is consistent with the server's networking configuration.
10) Interface names are arbitrary but usually are of the form **en***x*, **eth***x*, etc. One interface must be assigned a public address and the other the private address. This mechanism works best when all of the servers in a cluster use the same interface naming conventions; though the address assignments may differ. If this is not the case, you will need to use the **if-else-fi** configuration syntax to special case particular nodes.
11) Dual public/private network are fully supported in **Linux**, **MacOS**, and in **Solaris 11**.

Example

```
xrd.network nokeepalive nodnr
xrd.network wan buffsz 512k
```

3.2.1 Dual Public/Private Network Guidelines

Before you configure a dual public/private network, determine if such a network is actually necessary. Administering dual networks is difficult and problem resolution rather onerous. Typical reasons and alternatives to running a dual network are:

- Conserving IPV4 addressees: consider using IPV6, which does not suffer from this problem.
- Performance: if private addresses are routed along with public addresses over the same networking hardware then the switch becomes the bottleneck and performance improvements are moot.
- Security: if you desire to restrict certain kinds of access you may be able to achieve the same result using router and switch settings (e.g. file walls, routing restrictions, etc) thus avoiding a dual network.

If you still wish to run a dual network, you should use the following guidelines to avoid access surprises and mysterious performance issues.

- Never register private addresses in a publicly accessible DNS server. This exposes your private network configuration and is considered bad practice. Private addresses should only be registered in a private DNS or zone registered to prevent external leakage.
- Never register a server's private and public address under a single host name unless both addresses have equal connectivity. While this practice works for simple applications, complex applications like **XRootD** and **Proof** attempt to use all addresses assigned to a particular host name. If connectivity is unequal, performance issues and connection failures are likely to occur.
- Never cluster servers with only a public address with servers that only have a private address if their name spaces overlap. While this may work for certain network routing topologies, it invariably introduces inconsistencies.
- Carefully review the **routes** option on the **xrd.network** directive to make sure your network routing topology and interfaces are correctly stated.

3.2.2 Resolving Private IP Addresses

By default, private **IP** addresses are not resolved to host names. This is commonly accepted practice to avoid **DNS** timeouts as private addresses are usually not registered in **DNS**. However, some installations may opt to register such addresses in either a private **DNS** or in a zoned public **DNS**. This actually may be necessary in cloud deployments where nodes within the cloud receive private addresses and a **NAT** box is installed to provide outside access. In such cases, here is usually a 1-to1 mapping between a public address (e.g. **IPv6**) to a corresponding private address (e.g. **IPv4**). The public address is registered in a public **DNS** while the private address is registered in a private **DNS** within the cloud. Both addresses are registered with the same host name. However, to avoid leaking private addresses you must specify the **xrd.network rpipa** option so that only host names are returned and not actual addresses. This allows clients on the public network to accesses nodes in the private network using the **NAT** box. Nodes in the private network also receive host names but since these names are registered within the private network, proper address resolution occurs.

3.3 protocol

```
xrd.protocol [wan] name[:port] { lib | * } [parms]
```

Function

Configure a protocol that **xrd** is to use for incoming requests.

Parameters

wan Indicates that the protocol is able to effectively use the wan port for connections.

name The name of the protocol you wish to configure.

port The port number the protocol is to use for incoming requests. Specify a number, the name of a TCP service, or the word **any**.

lib The path to the shared library that contains the code the implements the protocol. If lib is an asterisk, the protocol has been statically linked with the daemon and should not be dynamically loaded.

parms Parameters to be passed to the protocol at load time.

.

Defaults

Not applicable.

Notes

1) The daemon expects that only one protocol is statically linked with the daemon's executable. The name of this protocol must correspond to the name of the program implementing the daemon (e.g., **xrootd**). You may configure this protocol using the **protocol** directive.
2) Additional protocols directives naming a non-default protocol include additional protocols. The daemon attempts to match each such protocol with the incoming connection in the order that the protocols are specified. The default protocol is always tried first, if applicable.
3) Additional protocols are dynamically loaded from the indicated *lib*.
4) If a port is not specified, then the standard port selection rules apply. Refer to the description of the **port** directive for port selection details.

5) A port of any assigns an arbitrary port number to the protocol.
6) Only those protocols bound to a specific port are matched against incoming connections on that port.
7) Load-time parameters are specific to each protocol. Refer to the protocol requirements for details.
8) The **xrootd** protocol does not need any load-time parameters.
9) The **xrootd** protocol is able to effectively handle the **wan** port. The **wan** port is an arbitrary port number that is automatically used by out-of-domain clients (i.e., an **xrootd** client inherently knows how to determine the actual port number). Typically, this port number has a much larger network buffer size to make up of wide area network latency.
10) Up to eight different protocols may be specified.

Example

```
xrd.protocol wan xrootd *
xrd.protocol xproofd:1093 /opt/xproofd/lib/libXProof.so
```

3.4 report

```
xrd.report dest1[,dest2] [every rsec] [-]option

option:  all | buff | info | link | poll | process |
         prot[ocols] | sched | sgen | sync | syncwp
         [[-]option]
```

Function

Specify execution tracing options.

Parameters

dest1 is a *host*:*port* or a UDP named local socket where reports are to be sent. Reports are always sent as a single UDP message.

dest2 is a secondary destination and must differ from *dest1*. The same report is delivered to *dest2* and *dest1*.

rsec determines how often reports are sent. Specify a number, optionally suffixed by **m** for **m**inutes, **s** for **s**econds (the default), or **h** for **h**ours. The default is every 10 minutes.

option Specifies the reporting 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. The following options produce reports on:

all	selects all possible reports
buff	I/O buffer activity
link	connection and socket I/O activity
poll	socket activity other than I/O
process	process resources
protocols	protocol specific information
sched	scheduling and thread activity
sgen	statistics generation
sync	synchronizes data for completeness (see the notes)
syncwp	synchronizes data only when practically possible

Defaults

Reporting is disabled.

Notes

1) Report messages are encoded in XML format. Refer to the **xrootd** protocol specification on the actual format and embedded information; as described under the response format for the **kXR_QStats** option of the **kXR_query** request.
2) By default, statistical values are obtained without data access synchronization. This may occasionally produce incomplete or inaccurate values. However, because information is collected asynchronously this has little impact on the server.
3) If absolute accuracy is required, you should specify the **sync** option. Be aware that reporting may require a significant amount of elapsed time while the server synchronizes its activities in order to produce accurate and consistent data.
4) If absolute accuracy is desired but not required, you should specify the **syncwp** option. The server synchronizes its activities only when possible. If the server is too active, an asynchronous report is done. The **sgen** report segment provides information on whether the report was synchronous or not and how much time it took to generate.
5) For scalability reasons, you should feed all UDP messages to one or more collectors whose sole function is to multiplex the UDP message streams into a single buffered serial stream. Generally, attempting to do more in a UDP message receiver substantially increases the chance for lost UDP messages.
6) An UDP message multiplexor and XML parser, **mpxstats**, is available as part of the reference xrootd distribution. Refer to the "Scalla Monitoring" reference for details.

Example

```
xrd.report myhost:1234 every 15m all -poll
```

3.5 sched

```
xrd.sched parms

parms:    [avlt avlt] [core {asis | max | off}]

          [idle idle] [maxt maxt] [mint mint] [stksz size]
```

Function

Specify when threads are created, how many can be created, and when they should be destroyed.

Parameters

avlt *avlt*

The number of threads that must always be available to service a request. These threads are never bound to any connection. Excess threads above this quantity will be allowed to bind with a socket until either the socket becomes idle or the number of available threads falls under *avlt*. The default is one half of the *mint* value.

core {**assis** | **max** | **off**}

Sets the limit for core file production. Choices are

assis - leave the current setting alone.

max - allow core files up the hard limit maximum (the default).

off - turn off core file production.

idle *idle*

The interval between checks for underused threads. Underused threads in excess of the *mint* value are terminated. Specify a number, optionally suffixed by **m** for **m**inutes, **s** for **s**econds (the default), or **h** for **h**ours. The default is every 13 minutes (i.e., 13m). Specifying a value of zero prevents threads from being terminated even if they are idle.

maxt *maxt*

The maximum number of threads that may be created to service requests. The number of threads will dynamically vary between *mint* and *maxt*.

mint *mint*

The minimum number of threads that must exist to handle requests. Once this number has been created, it is never reduced.

stksz *size*

The default thread stack size. Specify the number of bytes, optionally suffixed by **k** for **k**ilobytes or **m** for **m**egabytes. The default is controlled by the target operating system.

Defaults

```
xrd.sched mint 8 maxt 2048 avlt 512 idle 780
```

Notes

1) The *mint*-number of threads are eventually created.
2) The stack size is controlled by the target operating system used to create xrootd. For instance, in Solaris when sizeof(long) is 4 (indicating a 32-bit architecture), a 1M stack size is used. When sizeof(long) is 8 (indicating a 64-bit architecture or an LP64 data model) a 2M stack size is used. In Linux, the stack grows, as needed, to a maximum of 2M.
3) You should periodically review whether or not you have sufficient number of threads. The daemon prints a warning message the first time the *maxt* value is reached and more threads are needed.
4) If the daemon indicates that the thread limit was reached but less than *maxt* threads were created; then the target operating system maximum has been reached. This now becomes the new *maxt* value.
5) ***Warning,*** you should not change the thread parameters unless there is an overpowering reason to do so. The system is optimized for having many thread readily available. Constraining the number of threads may yield random failures that are hard to explain.

Example

```
xrd.sched mint 10 maxt 100 avlt 20
```

3.6 sitename

```
all.sitename sname
```

Function

Specify the site name to be included in monitoring records.

Parameters

sname

The 1- to 63-character name of the site. The name may include letters, digits, and the symbols "_-:.". Other symbols are converted to a period. If the name is longer than 63 characters it is truncated to 63 characters.

Defaults

None.

Notes

1) The **–S** command line option overrides the **sitename** directive.
2) The first **sitename** directive takes precedence over any subsequent **sitename** directives.

Example

all.sitename slac

3.7 timeout

```
xrd.timeout parms

parms: [hail hlto[h | m | s]] [idle idto[h | m | s]]
       [kill klto[h | m | s]] [read rdto[h | m | s]]
```

Function

Specify timeout parameters.

Parameters

hail *hlto*

The maximum number of seconds to wait for data to arrive after a connection is accepted. Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds, the default.

idle *idto*

The number of seconds a connection may remain idle before it is closed. Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds, the default. A value of 0 disables idle timeout processing.

kill *klto*

The number of seconds to wait for an "end session" request to complete. Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds, the default.

read *rdto*

The number of seconds a read may wait for data before it is either terminated or rescheduled. Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds, the default.

Defaults

```
xrd.timeout hail 30 idle 0 kill 3 read 5
```

Notes

1) The **idle** timeout prevents accumulation of dead connections which may happen when a client host machine crashes.
2) Currently, **idle** timeouts are disabled. You may enable them by specifying an *idto* value greater than zero.
3) Forced closure of connections is safe if the protocol supports dynamic reconnection, as the **xrootd** protocol does.
4) The **read** timeout forces a link to be closed should the initial protocol identification data not arrive within the timeout interval. After which, connections that do not send all of their data in the indicated period are simply rescheduled to the background.

Example

```
xrd.timeout idle 120m read 10
```

3.8 trace

```
xrd.trace [-]option

option:  {all | conn | debug | mem | net | none | off |
          poll | protocol | sched} [[-]option]
```

Function

Specify execution tracing options.

Parameters

option Specifies 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
conn	traces connection activity
debug	traces internal activities for debugging purposes
mem	traces memory management functions
net	traces network management functions
none	traces nothing
off	a synonym for **NONE**
poll	traces I/O interrupt polling activities
protocol	traces protocol activity (see the notes)
sched	traces scheduling functions

Defaults

Tracing is disabled.

Notes

1) All tracing is forcibly enabled when the daemon is invoked with the **–d** option.
2) All previous trace settings are discarded when **none** or **off** is encountered.
3) The **protocol** trace option is passed along to the all loaded protocols that may or may not respect the option or may have their own options.

Example

```
xrd.trace all -debug
```

4 Common xrootd Configuration Directives

4.1 export

```
all.export {path | *[?]} [[no]lock] [oss_options]
```

Function

Specify a valid path prefix for file requests.

Parameters

path An absolute path prefix for valid file requests. Only files starting with this prefix are allowed in requests.

***** Allow arbitrary object identifiers (i.e. names that do not start with a slash). The names are not inspected in any way and passed as is to the file system plugin.

***?** Allow arbitrary object identifiers (i.e. names that do not start with a slash). Inspected the names for CGI information and, if present, separate it from the object identifier (i.e. characters before the question mark) before passing the object name and CGI information to the file system plugin.

lock Uses standard xroot protection against multiple writers. This is the default.

nolock

Does not protect against multiple writers.

oss_options

Optional **oss** options that affect how the path is processed by the storage system and cluster service. Refer to the "Open File System & Open Storage System Configuration Reference" and the "Clustering Configuration Reference".

Defaults

```
xrootd.export /tmp lock
```

Notes

1) For security purposes, only files in **/tmp** are allowed to be accessed unless you specify otherwise. You may specify valid paths either on the command line or using the **export** configuration directive.
2) Do ***not*** prefix *path* with the **oss localroot** directive path, if any.
3) By default, a file may be opened by a single writer with no readers or multiple readers without any writer. If an external locking mechanism is used or no locking mechanism is needed; specify the **nolock** option to disable the default.
4) The [**no**]**lock** option *must* appear *before* any **oss** options.
5) The underlying file system plugin as well as the storage system plugin must support object identifiers in order to use the ***** or ***?** export. The default file system plugin will pass the object identifier to the storage system plugin for the common set of file operations. However, the default storage system plugin will not load if object identifiers are being exported.
6) Object identifiers exportation is meant to support object store plugins such as the Ceph block storage plugin.

Example

```
xrootd.export /store
```

4.2 seclib

```
xrootd.seclib {default | path}
```

Function

Specify the location of the security interface layer.

Parameters

default

Uses the default security plug-in the security interface.

path The absolute path to the shared library that contains an implementation of the Security (**sec**) interface that **xrootd** is to use for strong authentication(e.g., Kerberos, GSI, etc).

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.h**h for guideline on how to write a **sec** interface.
4) It is up to the **sfs** implementation to use authentication information to restrict access to files.
5) The provided **ofs** implementation can use authentication information for access control purposes.
6) The default **sfs** implementation does not provide any access control.

Example

```
xrootd.seclib /opt/xrootd/lib/libosec.so
```

5 Esoteric xrootd Configuration Directives

5.1 async

```
xrootd.async parms

parms: [force] [limit clim] [maxsegs smax]
       [maxstalls mstall] [maxtot slim]
       [minsz reqsz[k | m | g]] [minsfsz sfsz[k | m | g]]
       [off] [nosf] [segsz segsz[k | m | g]] [syncw]
```

Function

Specify how asynchronous I/O is to be handled.

Parameters

force Uses asynchronous I/O for all requests, even if the client did not ask for asynchronous handling.

limit *clim*

The maximum allowed number of outstanding asynchronous requests per client connection. Any additional requests past *clim* are synchronously handled. The default is eight (8).

maxsegs *smax*

The maximum number of simultaneous asynchronous operations that may any one request may have in progress. The default is eight (8).

maxstalls *mstall*

The maximum number of times a client may fail to deliver data at a sufficient rate to keep up with asynchronous I/O needs before future requests from the client are synchronously handled. Asynchronous handling is tried again after *mstall* number of synchronously handled requests. The default is eight (8).

maxtot *slim*

The maximum number of simultaneous asynchronous operations the server may have in progress. The default is 4,096.

minsz *reqsz*

The minimum number of bytes that must be read or written in a single client request for that request to be asynchronously handled. I/O requests smaller than *reqsz* are always synchronously handled. The *reqsz* can be suffixed by **k**, **m**, or **g** to indicate **k**ilo-, **m**ega-, or **g**iga-bytes; respectively. The default is one half of the initial *segsz*.

minsfsz *sfsz*

The minimum number of bytes that must be read in a single client request for that request to be handled using sendfile(). I/O requests smaller than *sfsz* are always handled in the standard way. The *sfsz* can be suffixed by **k**, **m**, or **g** to indicate **k**ilo-, **m**ega-, or **g**iga-bytes; respectively. The default 8192 for Linux, 1 otherwise.

off Disables asynchronous I/O for all requests.

nosf Disables using sendfile(), where available, for all read requests.

segsz *segsz*

The ideal asynchronous I/O segment size. The server attempts to quantize asynchronous I/O request into *segsz* pices. The *segsz* can be suffixed by **k**, **m**, or **g** to indicate **k**ilo-, **m**ega-, or **g**iga-bytes; respectively. The default is 64k. See the usage notes on how the system dynamically adjusts this value.

syncw

Uses synchronous I/O for all **fsync** requests. Otherwise, asynchronous I/O is used for **fsync** requests if the client requested asynchronous I/O or if the **force** has been specified.

Defaults

```
xrootd.async limit 8 maxsegs 8 maxstalls 5 maxtot 4096 segsz 64k
```

Notes

1) Asynchronous requests allow the client to start a number of read operations at one time and wait for the request to complete in optimal order. When properly employed, asynchronous requests may substantially improve overall client processing speed.
2) Asynchronous processing represents a substantial resource commitment on part of the daemon. Each operation requires the dispatching of a separate thread. Rampant asynchronous processing may exhaust resource limits. Use the **maxpl** and **maxps** values to limit the amount of asynchronous processing.
3) Asynchronous processing is effective when the disk transfer rate approaches the network transfer rate. Thus, asynchronous processing is enabled only when a sufficiently large amount of data is requested by the client at one time. Use the **minsz** parameter to control the point where asynchronous operation is effective.
4) The **segsz** parameter specifies the ideal i/o size for asynchronous operations in order to maintain continuous disk/network transfer overlap. Requests are broken into **segsz** units. If the requested size is smaller than **segsz**, the server will use half the **segsz**. If the number of units is more than twice **maxsegs**, the server will use twice the **segsz**.
5) Conversion of asynchronous requests to synchronous requests is transparent to the client.
6) The sendfile() interface allows data to be transferred directly from the kernel's file system memory cache to a client. Generally, this significantly reduced system overhead. The sendfile() function is available in Solaris 5.8 and above, as well as Linux 2.6 and above.
7) Use the **nosf** option in cases where you suspect that the sendfile() interface is causing data transfer problems.

Example

```
xrootd.async minsz 1M
```

5.2 chksum

```
xrootd.chksum [chkcgi] [max num] digest [path [args]]

digest: algorithim [digest]
```

Function

Specify how file check sums are computed.

Parameters

chkcgi

Always checks the **cgi** information, if any, for the **cks.type** element that can be used to select a checksum algorithm. The **cgi** information is not checked if only one digest is specified for backward compatibility. It is always checked if more than one digest is specified, making **chkcgi** unnecessary. See the usage notes for more information.

num Maximum number of checksum calculations that may run at the same time. Specifying 0 prevents real-time check summing. See the notes for more information.

algorithm

The name of the checksum digest (e.g., md5) used for the check summing. Specify one or more supported digests, each separated by a space. The first algorithm becomes the default algorithm. See the usage notes on how multiple digests are supported,

path The absolute path of the program that computes the check sum. If *path* is not specified, checksums are internally performed.

args Initial arguments to be passed to the program identified by *path*, if any.

Defaults

The default **max** is 4; otherwise. If *path* is not specified, checksums are internally performed. File check summing is not supported unless the directive is specified.

Notes

1) When a client issues an **xrootd** query checksum request, the following steps are performed:
   a. A checksum digest is selected as follows:
      i. If a single algorithm is specified and **chkcgi** was not specified, the digest in the configuration file is used.
      ii. If a single algorithm is specified and **chkgi** is specified, a **cgi** scan is made for **cks.type** and, if specified, its argument must match the single algorithm in the configuration file or an error results. In any case, the digest in the configuration file is used.
      iii. If more than one algorithm has been specified, a **cgi** scan is made for **cks.type** and, if specified, its argument must match one of the specified algorithms in the configuration file or an error results. If there is a match, the **cks.type** argument is used as the desired digest. If **cks.type** is not found, then the first algorithm specified in the configuration file is used.
   b. A check is made that the client has lookup privileges for the file and that a valid checksum has been recorded for the file. If both are true, that checksum is sent back to the client. If the client lacks lookup privileges, an access error is sent back to the client.
   c. Since a checksum needs to be computed the **max** value applies. If it is zero, the client is told that the checksum is not available.
   d. If the checksum is natively supported and no program path has been specified, a new checksum is locally computed and recorded for future queries. Otherwise, the program named in path is executed to compute a new checksum and it is not recorded for future queries. Of course, the program may record the checksum in some way for future queries.
   e. Either the previously recorded checksum or the computed checksum is provided to the client.
2) Native checksums are **adler32**, **crc32**, and **md5**.
3) Use the **ofs.ckslib** directive to add new digests or improve the performance of the native digests.
4) Since computation of multiple checksums is CPU and memory intensive choose the **max** with circumspection. You can control memory usage via the **ofs.cksrdsz** directive.
5) The **ofs** directives are documented on the OFS/OSS reference manual.
6) When the program identified by *path* is invoked, it is passed the path to the file that is to be processed. The actual argument list varies depending on whether or not a single algorithm has been specified, as follows:

   a. When a single algorithm has been specified, the program is passed the file path as the last argument and is the only argument if no *args* have been specified.
   b. When multiple algorithms have been specified, the program is passed the file path as the last argument and the checksum algorithm name as the second to the last argument. If no *args* have been specified, these are the only two arguments that are passed.

7) The program must output on standard out a single checksum value; normally ending with a new-line ('\n') character and terminate with a status code of zero. If the program terminates with a non-zero status code or returns no output, the client's request fails.
8) Upon success, the returned checksum value is passed back to the client, prefixed by the digest token, *digest*.
9) ***Warning***: If an external checksum program is specified (i.e. *path* is specified), then neither the **oss.localroot** nor **oss.namelib** directives are applied to the logical file name before passing the file name to the specified program that computes the checksum. Hence, the program is responsible for converting a logical file name to a physical file name.
10) When checksums are natively computed (i.e., path is not specified), then the **oss.localroot** and **oss.namelib** directives are applied to the logical file name. The checksum is computed against the resulting physical file name.
11) The **chkcgi** option is provided for backward compatibility. In previous releases only one algorithm could be specified and **cgi** information was immaterial. This processing mode remains the default when only a single algorithm is specified. You may wish to verify that a client as not requesting an unsupported digest in the case where some servers support multiple checksums and others do not.
12) The administrator's interface allows you to list and cancel checksum jobs. This applies to external as well as internal computation of the checksum.
13) When max is zero, checksums on demand are prohibited. This requires that checksums to be pre-computed. This can be done using the **frm_admin chksum** command. See the File Residency Manager reference.

Example

```
xrootd.chksum max 2 md5
```

5.3 diglib

```
xrootd.diglib * authpath
```

Function

Enable remote debugging via the **digFS** read-only file system.

Parameters

***** Loads the built-in version of **digFS**; the only the version currently supported.

authpath

The path to the authorization file that describes who is allowed to access **digFS** and what kind of information they may view.

Defaults

Bu default, **digFS** is disabled.

Notes

1) The **digFS** provides a virtual read-only file system view of key information about **xrootd** and **cmsd** that is valuable to remotely debug system problems.
2) Since **digFS** exposes system information an authorization file describing access permissions is required. See the next section.
3) When **diglib** is specified, the **/=/** directory is automatically exported and available to authorized users. You *must not* list **/=/** in the **all.export** list.
4) The /=/ path always refers to local storage regardless of server role and is never subject to redirection.
5) Only **close**, **dirlist**, **locate**, **open**, **read**, and **stat** requests can be vectored to **digFS**. Other requests referring to **/=/** are disallowed.
6) The **digFS** accepts configuration directives starting with **dig.** Refer to subsequent sections for a description of these directives.

Example

```
xrootd.diglib * /etc/xrootd/digauth.cf
```

5.3.1 Authorizing digFS Access

The file describing **digFS** access permissions is composed of newline delimited records. Each record describes a single entity that is authorized to access certain information. The format of each record is

```
info allow aprot ident

info:  all | [-]conf | [-]core | [-]logs | [-]proc | [info]

aprot: gsi | host | krb5 | pwd | sss | unix

ident: g=group | h=host | n=name | o=org | r=role | [ident]
```

Parameters

info Authorizes the entity described in the record to access certain information. Use the word **all** to allow access to all information. If you specify **all**, you can remove specific information by specifying subsequent information keywords prefixed by a minus sign. Alternatively, list the *info* keywords to enable access to the associated information described below.

Keyword	Information	Keyword	Information
conf	configuration file	**logs**	log files
core	core files	**proc**	process information (Linux only)

aprot Specifies the authentication protocol that must be used in order to use **digFS**. Only one protocol per entity description may be specified. Since the information provided by **digFS** is sensitive in nature you should use the strongest authentication protocol consistent with site policies. The following table lists the default protocols[1] from strongest to weakest. Additionally, the rightmost column lists the *ident* tags that can be successfully specified relative to that protocol since not every protocol identifies clients in the same way. See the **XRootD** security reference for more detailed information.

[1] Authentication is plugin based and any implemented authentication protocol may be specified as *aprot*.

Protocol	Descritption	Meaningful *ident* Codes
krb5	Kerberos Version 5	**h n**
gsi	Grid Security Infrastructure (i.e. x.509)	**g h n o r**
sss	Simple Shared Secret	**g h n**
pwd	Password	**g h n o r**
host	DNS resolved hostname	**h**
unix	NFS V2-Style authentication	**g h n**

ident Authorizes the *aprot* authenticated entity possessing the specified identity values access to info **digFS** information. Identity values are specified as key value pairs. The entity must match all the specified pairs in order to be granted access. An imbedded space in a value must be designated as a **\s** (i.e. two character sequence). Specifying an inappropriate key relative to an authentication protocol prohibits access. The following table describes the possible key value pairs.

Key	Value
g	Group name
h	Fully qualified hostname
n	Authentication-specific protocol client identity string (see notes)
o	Organization name
r	Role name

Notes

1) Valid entries in the *authfile* are used and syntactically incorrect entries are discarded. At least one valid entry must exist for **digFS** to be enabled.
2) If the modification time of the *authfile* changes outside of a 5 second window it is reprocessed. This allows you to modify the *authfile* on a running system. However, you must atomically update the file as follows:
   a. Create a copy of the file.
   b. Modify the copy as needed.
   c. Rename the copy to be the original name (i.e. use mv).

   Distributing a modified copy of the file to other hosts should also use rename to install the new *authfile*.

3) Each authentication protocol has a specific way of identifying a client. For instance, x.509 (i.e. **gsi**) uses distinguished name (i.e. dn). Depending on the security configuration the protocol-specific name may be mapped to a Unix name. If so, you must use the mapped name not the original name.
4) If an entity is associated with more than one group name then the specified group (i.e. **g=**) must match one of the associated group names.
5) Starting in version 4.2 you are able to enable **digFS** but prevent its use by simply not creating an authorization file or commenting out all authorization entries. This allows you to enable its use in real-time without restarting **XRootD** by either creating an authorization file or adding authorization lines to an existing file.
6) Prior to 4.2 you must have a valid authorization file with at least one authorization entry. However, that entry may be unsatisfiable. This also allows you to selective enable or disable **digFS** without an **XRootD** restart.

Example

```
all -core allow krb5 h=test.org n=xtestor
conf logs allow gsi g=atlas n=theuser
```

5.3.2 Optional digFS Directives

The **digFS** accepts the following directives in the configuration file.

5.3.2.1 addconf

```
dig.addconf path [ fname ]
```

Function

Add a configuration file reference to the **digFS** namespace.

Parameters

path The absolute path to a regular file that is to be added to "**/=/conf/etc**". The name of the file will be the same as the last component of path unless *fname* is specified.

fname The name that is to appear in "**/=/conf/etc**" but refers to path.

Defaults

None.

Notes

1) The *path* is only added if it is readable by the **xrootd** server.
2) This directive allows you to make other server related configuration files available via **digFS**.

Example

```
dig.addconf /etc/sysconfig/xrootd
```

5.3.2.2 log

```
dig.log parm [ parm ]

parm:    deny | grant | none
```

Function

Control the level of logging.

Parameters

parm The level of logging; specify one or more of:

deny - log file access denials
grant - log file access approvals
none - turn off logging

Defaults

```
dig.log deny grant
```

Notes

1) To enable logging of denials and approvals you must specify both **deny** and **grant** parameters.

Example

```
dig.log deny
```

5.3.3 Using digFS

The **digFS** file system can be accessed using standard file system applications. All information is rooted in the **/=/** directory and follows a standard layout. The following table describes directory tree.

Directory	Subdirectory	Contents
/=/conf		Configuration files **cmsd.cf** and **xrootd.cf**
	/etc	Other site selected configuration files.
/=/core		Core files
	/cmsd	Directory holding **cmsd** core files.
	/xrootd	Directory holding **xrootd** core files.
/=/logs		Log files
	/cmsd	Directory holding **cmsd** log files.
	/xrootd	Directory holding **xrootd** log files.
/=/proc		/proc files (Linux only)
	/cmsd	Directory holding the **cmsd** proc files.
	/xrootd	Directory holding the **xrootd** proc files.

If the **cmsd** and the **xrootd** share the same configuration file the **/=/conf/cmsd.cf** and **/=/conf/xrootd.cf** will be identical. If they share the core file directory or the log file directory; the same files may appear in the **cmsd** and **xrootd** subdirectories. As other components are added to **digFS**, additional executable names may appear in each root subdirectory.

To find out your access rights, simply list the entries in the **/=/** directory. Only the subdirectories for which you are authorized are displayed.

5.4 fslib

```
xrootd.fslib [throttle | [-2] path2] {default | [-2] path1}
```

Function

Specify the location of the file system interface layer.

Parameters

throttle

Loads "libXrdThrottle.so" to wrap the subsequent library specification.

path2 The path to the shared library that is to be used as the wrapper for the subsequent library specification.

-2 Uses version two of the interface for file system object instantiation.

default

Loads a built-in version of "libXrdOfs.so" as the file system implementation.

path1 The path to the shared library that contains an implementation of the Open File System (**ofs**) interface that **xrootd** is to use for file system specific operations (e.g., open, close, read, write, rename, etc).

Defaults

```
xrootd.fslib default
```

Notes

1) When you only specify the shared library filename, the library is located using the standard platform-dependent loader rules (e.g. well know places followed by the LD_LIBRARY_PATH envar setting).
2) The **sfs** interface allows you to provide an arbitrary file system implementation. It is documented in **XrdSfsInterface.hh**. Refer to this include file for differences between version 1 and 2 instantiation.

Example

```
xrootd.fslib /opt/xrootd/lib/libofs.so
```

5.5 fsoverload

```
xrootd.fsoverload [[no]bypass] [redirect target]
                  [stall sec]
target:   host:port[%host:port]
host:     dnsname | [ipv6addr] | ipv4addr
```

Function

Specify how to handle file system overload.

Parameters

[no]bypass

Specifies whether or not clients should be redirected to the client-specified forwarding destination when the file system indicates it is overloaded. This option is only meaningful for servers configured as forwarding proxies and is ignored if that is not the case. It is only applicable when the client specifies a forwarding destination.

redirect *target*

The name or address of the *host* and *port* number where clients are to be redirected when the file system indicates that it is overloaded. The target consists of one or two *host*:*port* specifications with the second separated by a percent sign (%). When a second *host*:*port* is specified, then clients connecting using a private IP address are redirected to the second *host*:*port* while clients connecting with a public IP address are redirected to the first *host*:*port*. If only one *host*:*port* is specified, all clients are redirected to that host.

stall *sec*

Specifies how long the client should be stalled when a redirect target is not available and the file system indicates that it is overloaded. After the stall, clients will re-issue the request. A value of zero passes back an overload error to the client when the file system indicates an overload.

Defaults

`xrootd.fsoverload nobypass stall 33`

Notes

1) The **fsoverload** directive is most effective for disk caching proxy servers. Refer to the "Proxy Storage Services Configuration Reference" for additional information on how to effectively use this directive.
2) Currently, the **fsoverload** directive only applies to file open requests. All other requests encountering a file system overload event fail with an overload error.
3) The **bypass** directive only applies to release 4.0 or higher clients. Older clients that specify a forwarding path are subject to the **stall** option should a file system overload event occur.

Example

```
xrootd.fsoverload bypass redirect foo.proxy.edu:1094
```

5.6 log

```
xrootd.log [-]levent [ [-]levent ] [• • •]
levent:    all | disc | login
```

Function

Specify event logging options.

Parameters

levent Specifies the events to be logged level. One ore more events may be specified. The specifications are cumulative and processed left to right. Each event may be optionally prefixed by a minus sign to turn off the setting. Valid events are:

all	logs all possible events, the default
disc	disconnect events
login	login events

Defaults

xrootd.log all

Notes

1) Events messages are routed to the **xrootd** log file.

Example

xrootd.log all -login

5.7 monitor

```
xrootd.monitor [ options ] dest [ dest ]

options: [all] [auth] [flush [io] intvl[m|s|h]]
         [fstat intvl[m|s|h] [lfn] [ops] [ssq] [xfr cnt]]
         [ident sec] [mbuff size[k] [rbuff size[k]]
         [rnums cnt] [window intvl[m|s|h]]
dest:    dest events host:port
events:  [files] [fstat] [io] [info] [redir] [user]
```

Function

Enable I/O monitoring.

Parameters

all Automatically enables monitoring for all connections. If **all** is not specified, monitoring is only enabled upon client request.

auth includes authentication information along with user information, when **user** is specified and authentication has been configured.

flush [**io**] *intvl*

The maximum time event data may be internally buffered before it is sent to the monitoring destination. Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds, the default. When **io** is specified, io event data is also subject to flushing. Otherwise, only non-io events are flushed. The default only applies to non-io events and is 10 minutes.

fstat *intvl* [**lfn**] [**ops**] [**ssq**] [**xfr** *cnt*]

Enables file activity monitoring using a special "**f**" stream. The *intvl* is the maximum time event data may be internally buffered before it is sent to the monitoring destination. Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds. A value of zero disables the "**f**" stream. The *intvl* is also used as the basis for **xfr** event data. By default, only file open and close events are inserted into the stream. Additional information may be requested as follows:

lfn includes the user's dictionary identifier along with the logical file name being opened in the open event record.

ops includes detailed operation count information along with minimum and maximum values in the close event record.

ssq includes the sum of squares count for read and write sizes in the close event record. Specifying **ssq** automatically includes **ops**. This option will impact server performance. See the notes for more information.

xfr *cnt* inserts the number of bytes read and written from each open file every *intvl***cnt* elapsed time. The *cnt* must be 1 or more.

ident *sec*

The number of seconds between each server identity transmissions (i.e., the '**=**' map record). Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds, the default. A value of zero transmits the identity only once at start-up time. The default is 1 hour (i.e. 3600 seconds).

mbuff *size*

The size of the monitoring datagram for file and I/O events. Specify no less than 1024 and no more than 64k as the maximum message size. The *size* can be suffixed by **k** to indicate **k**ilo-bytes. The default size is 16k.

rbuff *size*

The size of the monitoring datagram for redirection events. Specify no less than 2048 and no more than 64k as the maximum message size. The *size* can be suffixed by **k** to indicate **k**ilo-bytes. The default size is 32k.

rnums *cnt*

The number of redirection monitoring streams to start. Specify no less than 1 and no more than 8. The default size is 3.

window *intvl*

The monitoring window size. Data collected within the window is not differentiated by time. Thus, the window represents the undifferentiated sampling interval. Specify a number optionally suffixed by **h** for hours, **m** for minutes, or **s** for seconds, the default. The default is 60 seconds.

dest *host*:*port*

The name of the host where monitoring messages should be sent. The receiving port number must be specified after the colon. All monitoring messages are sent as datagrams (i.e., UDP protocol). The **dest** parameter must be specified as the last parameter. Up two destinations are allowed. By default, only file event information is sent. The actual events may be specified after the dest keyword. These events are:

files file-related request monitoring (i.e., open and close requests).
fstat the specified "f" stream information (i.e., open and close requests).
io I/O request monitoring (read and write requests plus files).
iov same as **io** above but also include details on **readv** vector elements.
info client specified monitoring data submitted using xrootd protocol.
redir redirection events.
user client login and disconnect events.

Defaults

While `flush 10m ident 1h mbuff 8k rbuff 32k rnums 3 window 60` is in effect; monitoring is not enabled.

Notes

1) Use the **monitor** directive to enable statistical gathering of file event and I/O requests.
2) The **fstat ssq** option impacts performance since floating point operations must be carried out for each read and write request in order to accurately compute the sum of squares. The counts can be used to compute the standard deviation for read and write sizes. Do not specify this option unless there is a clear need for such information.
3) The **fstat ssq** counts are available on platforms that use IEEE 754 floating point format. The **fstat ssq** option is ignored on non-conforming platforms.

4) The **io** option reports individual seeks for each read and write request. While this data may be used to determine access patterns or used in I/O trace simulation studies, it reduces server performance by about 7% and generates a large amount of monitoring data. Normally, **fstat** provides sufficient information about client I/O efficiency at a much lower cost.
5) The **flush** parameter does *not* apply to monitor streams that include **io** event data unless **io** is specified. By default, monitor streams that include **io** event data are flushed only when the internal monitor buffer becomes full or when the user owning the stream being monitored disconnects.
6) You may specify two monitoring destinations. This allows you to isolate data high volume streams (i.e., **io** monitoring) and provide real-time display for low-volume streams (i.e., **info**, **files**, **fstat**, and **user**).
7) The **all** option forces monitor data to be collected for all connections. If **all** is not specified, each client must enable monitoring manually using the **xrootd set** request code (see the **xrootd** protocol specification). This allows selective monitoring and gives each client the opportunity to tag **io** monitor data with the relevant application name.
8) The **iov** option inserts a read entry for every element in a **readv** vector. This may explode the amount of monitoring information that is generated. By default, when only **io** is specified, a summary **readv** entry is placed in the monitoring stream.
9) Clients cannot enable monitoring that has not been enabled by the **monitor** directive.
10) Specifying a small datagram buffer size (e.g. less than 8k) increases the number of datagrams that need to be sent and, consequently, adds to server overhead. Large datagram buffer sizes reduce the number of datagrams as well as server overhead but increase memory utilization as each connection allocates a buffer.
11) Approximately 61 requests can fit into a 1K **mbuff**.
12) On average, 64 to 128 redirection events can fit into a 32K **rbuff**.
13) Increasing the number of redirection monitoring streams (**rnums**) reduces the bottlenecks in the monitoring path.
14) Specifying a small window increases the timing accuracy of any individual request entry at the expense of additional datagrams and significantly increased server overhead. Conversely, large window sizes reduce timing accuracy but also reduce server overhead.

15) Refer to the "Scalla Monitoring" reference for a detailed explanation on the datagram format used by the monitoring subsystem. It is especially important to understand the different between **files**, **fstat**, and **io** monitoring as the information overlaps and there is rarely a need to specify all three.

Example

```
xrootd.monitor all fstat 5m dest fstat datacoll:5050
```

5.8 pidpath

```
all.pidpath path
```

Function

Specify the location of the **xrootd.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 location of the pid file is modified by the **–n** option.
2) The all prefix indicates that all components creating a pid file should place the file in the same location. To create a exception pid file location, use '**xrootd**' as the prefix instead of '**all**'.

Example

```
all.pidpath /var/run/scalla
```

5.9 prep

```
xrootd.prep parms

parms:    [ keep ksec ] [ scrub time ] [ logdir ldir ]
```

Function

Specify how prepare request tracking is done.

Parameters

keep *ksec*

The time that prepare request tracking record are to be held. The time may be suffixed by **s** (the default), **m** , or **h** to indicate seconds, minutes, and hours, respectively. The default is 24 hours.

scrub *time*

The time between scrubs of the tracking log directory. The time may be suffixed by **s** (the default), **m** , or **h** to indicate seconds, minutes, and hours, respectively. The default is 1 hour.

logdir *ldir*

The absolute path of the directory that is to hold the preparation tracking records. A directory must be specified, otherwise preparation request tracking is disabled.

Defaults

None. Preparation request tracking is normally disabled. When a **logdir** directory is specified, the **keep** and **scrub** defaults of **24H** and **1H** apply, respectively.

Notes

1) This directive allows server to track prepare requests. When request tracking is enabled, each prepare is logged in the **logdir** directory. It then becomes possible to list the requests and cancel them, if need be.

2) Since there can be more than one redirecting **xrootd** server, prepare requests may be scattered across several servers. It is the client's responsibility to collect information from each server in order to create a composite preparation request history.
3) Each server uniquely names the files in the **logdir** directory. When multiple **xrootd** redirecting servers exist, it is possible to collect full preparation history from any server, if the **logdir** directory is located in a shared file system (e.g., **NFS**).
4) When running multiple **xrootd** servers on the same machine, the instance name (**-n** command line option) is used to differentiate **logdir** directories among all instances by appending the instance name to the path.

Example

```
xrootd.prep keep 12H logdir /nfs/xrootd/preplog
```

5.10 redirect

```
xrootd.redirect target {[-]foper | [?] path [path […]]}
foper:    {all | chmod | chksum | dirlist | locate | mkdir
          | mv | prepare | prepstage | rm | rmdir | stat
          | trunc} [[-]foper]
target:   host:port[%host:port]
host:     dnsname | [ipv6addr] | ipv4addr
```

Function

Specify request forwarding.

Parameters

target The name or address of the *host* and *port* number where clients are to be redirected based on the subsequent parameters. The target consists of one or two *host*:*port* specifications with the second separated by a percent sign (%). When a second *host*:*port* is specified, then clients connecting using a private IP address are redirected to the second *host*:*port* while clients connecting with a public IP address are redirected to the first *host*:*port*. If only one *host*:*port* is specified, all clients are redirected to that host.

foper Specifies which metadata operations are to be *immediately* redirected. One or more operations may be specified. The specifications are cumulative and processed left to right. Each operation may be optionally prefixed by a minus sign to turn off the setting. Valid operations are:

all	redirect all possible operations
chmod	redirect change mode requests
chksum	redirect checksum requests
dirlist	redirect directory content listing requests
locate	redirect path location requests
mkdir	redirect create directory requests
mv	redirect rename requests
prepare	redirect prepare requests that *do not need* file staging
prepstage	redirect prepare requests that *may need* file staging

rm	redirect file removal requests
rmdir	redirect directory removal requests
stat	redirect file attribute requests
trunc	redirect file truncate requests using a path

path Specifies that when a file open request occurs on the specified path prefix, the client should be redirected to the specified host and port. One or more paths may be specified. However, no more than four different host-port combinations may be specified.

? *path* Specifies that any client operation on the specified path prefix that ends with a "not found" error (i.e., EONOENT) and has not been specifically covered by another redirect directive, the client should be redirected to the specified host and port. All subsequently specified paths, if any, on the line fall under the "not found" provision.

Defaults

```
xrootd.redirect -all
```

Notes

4) Request redirection is typically applicable to the cluster manager. Refer to the **role** directive in the "Clustering Configuration Reference" for additional information, especially on inter-related directives.
5) Normally, meta-data requests are performed on the local host. However, certain clustered environments may be controlled by a central manager that records the exact state of every file. In such environments, the central manager may perform meta-data requests. When the **redirect** directive is not specified, the client is directed to perform the operation on a single host, normally the one that has the file. When the request is redirected, the target host is responsible for performing the operation.
6) The redirect path prefixes are always matched from most- to least-specific prefix (i.e., longest to shortest).

Example

```
xrootd.redirect all -prepare
```

5.11 trace

```
xrootd.trace [-]option [ [-]option ] [• • •]
option:   all | debug | emsg | fs | login | mem | none |
          off | stall | redirect | request | response
```

Function

Specify execution tracing options.

Parameters

option The tracing level. One ore 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
debug	traces internal activities for debugging purposes
emsg	traces errors sent back to the client
fs	traces file system requests
login	traces login and authentication steps
mem	traces memory management functions
none	traces nothing
off	a synonym for **none**
stall	traces client deferrals due to resource limitations
redirect	traces client redirections to other servers
request	traces client request information
response	traces request response information

Defaults

Tracing is disabled.

Notes

1) All tracing is enabled when the daemon is invoked with the **–d** option.
2) All previous trace settings are discarded when **none** or **off** is encountered.

Example

```
xrd.trace all -debug
```

6 Enabling HTTP Access

XRootD supports **HTTP** access via a protocol plugin. The **HTTP** protocol can run alongside of standard **XRootD** protocol without any interference; providing an additional access mode. To Enable **HTTP** access, add the following configuration parameter to the configuration file.

```
if exec xrootd
xrd.protocol http[:port] path/libXrdHttp.so [cfgfile]
fi
```

Parameters

port The port number http is to use for incoming requests. Specify a number, the name of a **TCP** service, or the word **any**. If you do not specify port number, port 1094 is used. This is also the default port for **XRootD** protocol.

path The path to the shared library **libXrdHttp.so** that contains the code the implements the **HTTP** protocol.

cfgfile is the path to an external configuration file specific to **HTTP**. If not specified, the configuration file at daemon start-up is used.

Defaults

Not applicable.

Notes

1) You should surround the **xrd.protocol** directive with the shown **if**-**fi** if you are using a common configuration file for **xrootd** and **cmsd** daemons. Failure to do so will prevent the **cmsd** from starting.
2) All **HTTP** requests are bound by any "**xrd.**" and "**xrootd.**" Directives that exist in the start-up configuration file. Hence, **HTTP** access cannot exceed any restriction imposed by those directives.
3) Monitoring of **HTTP** requests is handled as if they are **XRootD** requests. Consequently, the monitoring stream includes all **HTTP** requests as well. However, monitoring information is tagged with the fact that the information was generated by the **HTTP** protocol.

4) While the default port number is the same as for **XRootD**; the framework directs each request to appropriate protocol and these requests are never intermixed. To avoid any confusion you may wish to use a more standard **HTTP** port number such as 8080. This is especially true if connection are made using **HTTPS**.
5) Additional configuration file directives specific to **HTTP** are always prefixed with "**http.**" and the following sections describe these directives.
6) **HTTP** support also includes **WebDav** support enabling a wider range of access abilities beyond simple get and post capabilities.
7) When **HTTP** is used in a clustered **XRootD** deployment, all servers in that deployment must have **HTTP** enabled. Failure to do so typically results in access failure when a client is redirected to a serer that holds the desired file but for which **HTTP** was not enabled.
8) Not all **HTTP** clients support all **HTTP** features. While the plugin does not violate the **HTTP** or **WebDav** standards, it does implement a wide range of allowable features (e.g. redirection on POST requests) that may not be supported by the **HTTP** client being used.

6.1 Enabling HTTPS

When a server is configured to use **HTTPS**, each server processes the client's credentials from the connection. This allows the client to be authenticated and makes authorization possible. On the other hand, **HTTPS** is very resource demanding because it encrypts and decrypts all of the **TCP** traffic. Additionally, the process of establishing an encrypted connection requires several network inter-changes that increase connection latency which can be substantial on a wide area network.

While **HTTP** is much faster it is impossible to authenticate the client using **HTTP**. However, the **HTTP** plugin allows a client to initially connect with **HTTPS**, extract the authentication information, encode that information in a low-overhead encrypted security token and redirect the connection to use **HTTP**. This is known as **HTTPS** to **HTTP** conversion and is much less resource intensive. When a server is configured to do **HTTPS** to **HTTP** conversion, it always expects a security token to be present when a client connects via **HTTP**. If the token is missing or cannot be decrypted the connection is rejected. This mechanism provides a relatively secure authentication but at the expense of privacy as no traffic is encrypted past the authentication stage.

HTTPS to **HTTP** conversion is especially attractive in clustered environments where a client typically makes contact with a particular node (i.e. redirector) that then redirects the client to a particular server that holds the requested file. Using **HTTPS** everywhere incurs identical overhead at each contact point. This can be eliminated by using **HTTPS** at the initial contact point and converting **HTTPS** to **HTTP** for subsequent connections. This incurs the overhead just once.

When enabling **HTTPS** you should consider the following points:

- If only **HTTPS** is configured, then the server only accepts **HTTPS** connections (see the **cadir**, **cafile**, **cert** and **key** directives).
- If **HTTPS** to **HTTP** conversion is configured a server accepts an **HTTPS** connection or an **HTTP** connection that provides a valid security token (see the **secretkey** directive)
- If self-conversion of **HTTPS** to **HTTP** is configured, the server unconditionally redirects any **HTTPS** incoming connections to itself; using **HTTP** and a security token (see the **selfhttps2http** directive). This allows greater performance for subsequent requests.

When using **HTTP** in an **XRootD** cluster, additional considerations apply on how the cluster redirector interacts with data servers in that cluster. The following table provides reasonable possibilities, depending on the degree of security that is desired.

Redirector Accepts	**Server Accepts**	**Configuration**	**Remarks**
HTTP	HTTP	The is the default	No security.
HTTPS	HTTP with security token	Specify **cadir**, **cert**, **key**, and **secretkey** directives.	Central authentication, fast unencrypted data access but higher CPU load in the redirector
HTTP	HTTPS	Specify **cadir**, **cert**, **key** directives *only* in data servers. Specify **desthttps** in redirectors.	Fast redirection, distributed authentication, slow encrypted data access
HTTP	HTTPS with self redirection using HTTP with security token	Specify **cadir**, **cert**, **key**, **selfhttps2http** and **secretkey** directives *only* in data servers. Specify **desthttps** in redirectors.	Fast redirection, distributed authentication, fast unencrypted data access
HTTPS	HTTPS	Specify **cadir**, **cert**, **key** directives in servers and redirectors.	Fully authenticated but authentication occurs twice, slow encrypted data access, resource consumption can be high

6.2 Directives to Enable HTTPS Access

By default, **HTTPS** access is not enabled. You must specify certain critical information in order for **HTTPS** to be enabled. The following sections describe these directives.

6.2.1 **cadir** *(required or use **cafile**)*

```
http.cadir path
```

Function

Specify the directory containing the CA certificates (see the **cafile** directive as an alternative).

Parameters

path The path to the directory.

Defaults

None.

Notes

1) All of the certificates in the directory must in a format that is recognized by the version of **OpenSSL** is being used.
2) If the certificate information is contained in a single file, you must use the **cafile** directive.

Example

```
http.cadir /etc/grid-security/certificates
```

6.2.2 **cafile** *(required or use **cadir**)*

```
http.cafile path
```

Function

Specify the file containing the CA certificates.

Parameters

path The path to the file.

Defaults

None.

Notes

1) All of the certificates in the file must in a format that is recognized by the version of **OpenSSL** is being used.
2) If certificates are contained in multiple files you must use the **cadir** directive.

Example

```
http.cafile /etc/myCA.pem
```

6.2.3 **cert** *(optional)*

```
http.cert path
```

Function

Specify the file containing the x.509 certificate that the server must use.

Parameters

path The path to the file.

Defaults

None.

Notes

1) The certificate must be in **PEM** format.
2) See the related **key** directive to specify the location of the private key.

Example

```
http.cert /etc/grid-security/hostcert.pem
```

6.2.4 desthttps

```
http.desthttps {no | yes}
```

Function

Specify whether or not **HTTPS** is to be used for redirections.

Parameters

no A redirector will always redirect a client using http. The word **false** and the number **0** are synonyms.

yes A redirector will always redirect a client using https. The word **true** and the number **1** are synonyms.

Defaults

http.desthttps no

Notes

1) While this directive applies normally applies to redirectors it is used by any node, redirector or data server that redirects a client.

Example

```
http.desthttps yes
```

6.2.5 **gridmap** *(optional)*

```
http.gridmap path
```

Function

Specify the file containing the "grid map file" that the server must use.

Parameters

path The path to the file.

Defaults

None.

Notes

1) This file is loaded at startup and used to translate the requestor's .X509 DN into a short user name for internal authorization usage.
2) See the related **cert** directive to specify the location of the server's certificate.

Example

```
http.gridmap /etc/grid-security/mapfile
```

6.2.6 **header2cgi** *(optional)*

```
http.header2cgi hdrkey cgikey
```

Function

Specify which headers are to be promoted to **cgi** information and appended to the incoming **url**.

Parameters

hdrkey the header key that is to be promoted to **cgi** information.

cgirkey the cgi key name that the promoted header should have.

Defaults

None.

Notes

1) Normally, header information is internally processed and not made available to other plug-ins. The **header2cgi** directive allows you to pass on header information to external plug-ins via the incoming **url** by promoting the header payload to a **cgi** element.
2) Assuming *xyzzy* is the payload of header with a key of **auth**, the example shown below would promote the header by appending "**authz=***xyzzy*" to the incoming **url** as **cgi** information before it is passed to other system components. This essentially makes the header visible outside of the **http** plug-in.
3) The **header2cgi** directive is meant to be used for non-**http** plug-ins that wish to consider specific information sent via the **http** protocol.

Example

```
http.header2cgi auth authz
```

6.2.7 **key** *(optional)*

```
http.key path
```

Function

Specify the file containing the x.509 private key that the server must use.

Parameters

path The path to the file.

Defaults

None.

Notes

1) The key must be in **PEM** format.
2) See the related **cert** directive to specify the location of the server's certificate.

Example

```
http.cafile /etc/grid-security/hostkey.pem
```

6.2.8 secretkey

```
http.secretkey {path | token}
```

Function

Specify the key to be used to encrypt and decrypt redirection tokens.

Parameters

path The absolute path to the file containing a random string of alpha-numeric characters and symbols that are to be used to encrypt and decrypt redirection tokens.

token A random string of alpha-numeric characters and symbols that are to be used to encrypt and decrypt redirection tokens. The token may not start with a slash.

Defaults

None.

Notes

1) The same key must be used by all nodes within a cluster.
2) Specifying a secret key automatically enables **HTTPS** to **HTTP** conversion.

Example

```
http.secretkey 1agq45jkl3400vfghtqzz963
```

6.2.9 selfhttps2http

```
http.selfhttps2http {no | yes}
```

Function

Specify whether or not a server may redirect an **HTTPS** connection to itself using **HTTP** plus a security token.

Parameters

no A server should continue to use **HTTPS** for all communications. The word **false** and the number **0** are synonyms.

yes A server should convert an **HTTPS** session to an **HTTP** session by redirecting the client to itself using **HTTP** plus a security token. The word **true** and the number **1** are synonyms. You must also specify the **secretkey** directive.

Defaults

http.selfhttps2http no

Notes

1) This option is meant to control the level of data privacy that is desired. Normally, **HTTPS** connections are converted to **HTTP** connections after authentication information is extracted from the **HTTP** stream. This greatly reduces overhead as no data past the authentication stage has to be encrypted.
2) When an **HTTPS** connection is converted to an **HTTP** connection, the redirection includes a security token encrypted with the key specified by with the **secretkey** directive. The **HTTP** connection is only accepted if the token can be decrypted using the same key.

Example

```
http.selfhttps2http yes
```

6.2.10 secxtractor

```
http.secxtractor path
```

Function

Specify the location of the specialized authentication information extractor plugin.

Parameters

path The path to the shared library containing the plugin.

Defaults

None.

Notes

1) A Security eXtractor plugin is a component that can be loaded at initialization time in order to provide specialized processing to the certificate passed by the client. Normally, all authentication information comes from the standard part of the client certificate and any extensions are ignored.. A Security eXtractor can be used to extract other information from the certificate or any of its extensions. This information is then passed along and may be used for other authorization functions within **XRootD**.
2) The typical case for which a security extractor library is needed is to extract the extended VO information from a Grid client's certificate.

Example

```
http.secxtractor /usr/lib64/libXrdHttpVOMS.so
```

6.3 Common Directives

6.3.1 embeddedstatic

```
http.embeddedstatic {no | yes}
```

Function

Specify where **CSS** template and logo is to come from for formatted listings.

Parameters

no The CSS template and logo information must be over-ridden by another file containing such information. The word **false** and the number **0** are synonyms.

yes An internal memory-based CSS template and logo should be used. The word **true** and the number **1** are synonyms.

Defaults

http.embeddedstatic yes

Notes

1) Using the default memory-based CSS template and logo provide much better performance and makes the setup much simpler.
2) If you need to use a custom style sheet, significant performance gains can be achieved by preloading the style sheet file using the **staticpreload** directive.

Example

```
http.embeddedstatic yes
```

6.3.2 listingdeny

```
http.listingdeny {no | yes}
```

Function

Specify whether or not directory listings are allowed.

Parameters

no Directory listings are not allowed. The word **false** and the number **0** are synonyms.

yes Directory listings are allowed. The word **true** and the number **1** are synonyms.

Defaults

http.listingdeny no

Notes

1) In a clustered environment a listing of a directory via **HTTP** only lists the directory of some arbitrary server in the cluster. Since files are scattered across all of the servers in the cluster; this likely produces an incomplete listing. You may wish to deny directory listings to avoid confusion.
2) Alternatively, you can redirect directory listings to a special node that can produce a composite listing using all nodes in the cluster via the **listingredir** directive.
3) Note that the **XRootD** based **xrdfs** command automatically produces a composite listing.

Example

```
http.listingdeny yes
```

6.3.3 listingredir

```
http.listingredir desturl
```

Function

Specify the node to which to redirect clients requesting a directory listing.

Parameters

desturl The redirection URL to use when a directory listing is requested.

Defaults

None.

Notes

None.

Example

```
http.listingredir http://hostwhichprovideslistings:80/
```

6.3.4 staticpreload

```
http.staticpreload url path
```

Function

Preload a static resource file into memory.

Parameters

url The URL naming a static resource (e.g. style sheet or icon).

path The path to the local file containing the resource.

Defaults

None.

Notes

1) Static resources named in a URL must start with "/static" in order to be recognized.
2) The contents of the static resource contained in path may not exceed 64K. If it does, it is truncated to 64K.
3) Typical static resources are URL resources ending with ".css" and ".ico".
4) This directive is ineffective if the **staticredir** directive is specified.

Example

```
http.staticpreload http://static/mycss.css /etc/mycss
```

6.3.5 staticredir

```
http.staticredir newurl
```

Function

Preload a static resource file into memory.

Parameters

newurl The URL the client is to be redirected to when requesting a non-local or unsupported static resource.

Defaults

None.

Notes

1) The **staticredir** directive is only effective when a) **embeddedstatic** processing is disabled, or b) the resource is neither a content style sheet nor an icon.

Example

```
http.staticredir http://althost/
```

6.3.6 trace

```
http.trace [-]option [ [-]option ] [• • •]
option:   all | debug | emsg | fs | login | mem | none |
          off | stall | redirect | request | response
```

Function

Specify execution tracing options.

Parameters

option The tracing level. One ore 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
debug	traces internal activities for debugging purposes
emsg	traces errors sent back to the client
fs	traces file system requests
login	traces login and authentication steps
mem	traces memory management functions
none	traces nothing
off	a synonym for **none**
stall	traces client deferrals due to resource limitations
redirect	traces client redirections to other servers
request	traces client request information
response	traces request response information

Defaults

Tracing is disabled.

Notes

1) All tracing is enabled when the daemon is invoked with the **–d** option.
2) All previous trace settings are discarded when **none** or **off** is encountered.

Example

```
http.trace all -debug
```

7 Document Change History

14 March 2005

- Remove documentation on local redirection mode.
- Remove documentation of **–s** command line option.
- Add '**-t**' option to the StartXRD documentation.
- Significantly change the **port** directive, adding "**port any**" and "**if**".
- Discuss using "**port any**" mode.

26 April 2005

- Further clarified the **xrootd monitor flush** parameter.

1 June 2005

- Added description of conditional directives (**if**-**fi**).
- Added description of the **–n** command line option.
- Fully explain which run-time files are created.
- Deprecate **–r**, **–t**, and **–y** command line options.
- Deprecate the **XRDMODE** variable and remove the description of the **XRDTYPE** variable in the **StartXRD.cf** script.
- Remove extraneous options from the **StartXRD** script.

1 Aug 2005

- Document administrative interface portal socket.
- Add file size to open monitor record.

16 Aug 2005

- Add authentication mapping (**a**-record) to monitoring data.

6 Jan 2006

- Document the **-b** and **-R** command line options.
- Document how to independently bind different port numbers to available protocols.

25 Jan 2006

- Add max option to **chksum** directive.

22 March 2006

- Add **exec** condition to **if**/**else**/**fi**.

28 February 2007

- Cleaned up documentation relative to **role** directive and **all** prefix modifier.
- Documented the **xrootd.redirect** directive.
- Removed the **xrd.connections** directive.
- Placed most **xrd** directives in esoteric status.

28 March 2007

- Move conditional directives to a separate manual.
- Indicate the **adminpath** now is configured via the **all** prefix.
- Documented the xrd **wan network** and **protocol** directive option.
- Indicate that the **xrootd export** directive is configured via the **all** prefix and accepts **oss** options.

01 October 2007

- Document the **locate** option of the **redirect** directive.

01 January 2008

- Remove references to olbd.

01 February 2008

- General clean-up.

11 April 2008

- Document staging ('**s**') monitor record.

29 May 2008

- Document the **xrootd async nosf** option.

21 July 2008

- Document the **xrd network [no]dnr** option.
- Document the **xrd async minsfsz** option.

6 March 2009

- Document the **xrootd monitor stage** option.

22 June 2009

- Document the **xrd.report** directive.

7 July 2009

- Document the **mpxstats** command for monitoring.
- Document the summary variables.
-

17 March 2010

- Document the **timeout hail** and **kill** options.
- Document the pid file creation and the **pidpath** directive.

8 March 2011

- Document the **–s** command line option.
- Minor editorial changes.

24 May 2011

- Document the **auth** option in the **xrootd.monitor** directive.

31 May 2011

- Change the **xrootd.chksum** directive to support native checksums. Additional wording added explaining native checksums.

29 June 2011

- Document the **rbuff** and **redir** options on the **xrootd.monitor** directive to support redirection monitoring.

27 September 2011

- Document the **io flush** option on the **xrootd.monitor** directive.

---------- Release 3.1.0

10 October 2011

- Document the **iov, migr,** and **purge** options on the **xrootd.monitor** directive.

2 November 2011

- Update documentation on the **xrootd.redirect** directive. It now accepts additional file operations (**chksum** and **trunc**), **open** targets (previously undocumented feature), and **ENOENT** targets.

3 December 2011

- Remove the **migr**, **purge** and **stage** options from the **xrootd.monitor** directive. These have been moved to the **frm.all.monitor** directive.
- Document the new **ident** option on the **xrootd.monitor** directive.

12 December 2011

- Document the **rnums** option for the **xrootd.monitor** directive.

---------- Release 3.2.0
---------- Release 3.2.1
---------- Release 3.2.2
---------- Release 3.2.3
---------- Release 3.2.4

21 September 2012

- Document the **fstat** option for the **xrootd.monitor** directive.
- Remove the **rootd** configuration section.

---------- Release 3.2.5

22 October 2012

- Document the **–S** command line option and the **all.sitename** directive for specifying a monitoring site name.

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

15 December 2012

- Change the **fstat sdv** option to **fstat ssq** in the **xrootd.monitor** 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

11 February 2013

- Enhance the **fslib** directive to allow one to easily wrap one library with another.

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**, **–l** and **–z** command line options.
- Document exported environment variables.
- Document the environment information file contents.
- General clean-up and better explanations.

2 December 2013

- Document the **xrootd.diglib** directive.

8 January 2014

- Document the **routes** option on the **xrd.network** directive.
- Document enhanced **xrootd.redirect** directive that can distinguish between public and private IP addresses.

18 February 2014

- Restrict the **routes** option on the **xrd.network** directive to prohibit auto-discovery of interface addresses as this may lead to choosing the wrong addresses.

27 March 2014

- Document how to enable **HTTP** and **HTTPS** protocols.
- Redesign the **routes** option on the **xrd.network** directive to cover the most common case.

6 August 2014

- Document the core option in the **xrd.sched** directive.
- Document how to export object identifier names via the **all.export** directive.

8 September 2014

- Document that **TCP keepalive** is now the default setting.
- Add a **nokeepalive** option and a **kaparms** option to the **xrd.network** directive.
- Indicate the **use** option in **xrd.network** accepts one or two interface names.
- Minor corrections to HTTP section.
- Document the **http.mapfile**, **http.staticredir**, **http.staticpreload**, and the **http.trace** directives.

27 September 2014

- Document multiple checksum support via the **xrootd.chksum** directive.
- Document the **default** option on the **xrootd.seclib** directive.
- Document the **–L** command line option.

26 November 2014

- Document the version option in the **xrootd.fslib** directive.

10 February 2015

- Document how to pass command line arguments to plug-ins.
- Document how to enable **digFS** but prevent its use until needed.

25 November 2015

- Explain the side-effects of the **–s** command line option on the placement of the environmental file.

15 April 2016

- Document log file plug-ins.

20 June 2016

- Document the **cse** parameter for logging plug-ins.
- Document the **xrd.network** [**no**]**rpipa** option.

10 March 2017

- Correct **http.mapfile** directive (it's really **gridmap**).

20 May 2017

- Document the **xrootd.fsoverload** directive.

27 October 2017

- Document the **http.header2cgi** directive.