



# Scalla

## Xrd Configuration Reference

12 December 2011

Andrew Hanushevsky



Scalla: Structured Cluster Architecture for Low Latency Access  
©2004-2011 by the Board of Trustees of the Leland Stanford, Jr., University  
All Rights Reserved  
Produced under contract DE-AC02-76-SFO0515 with the Department of Energy  
This code is available under a BSD-style license allowing minimally restricted use.

<b>1</b>	<b>Introduction .....</b>	<b>5</b>
1.1	Security Considerations .....	6
1.2	Files Created by xrootd .....	6
1.3	Starting the xrootd Daemon .....	7
1.3.1	Using the StartXRD Script to Start xrootd .....	11
1.3.1.1	StartXrd.cf Configuration File .....	11
1.3.2	Using the StoptXRD Script to Stop xrootd .....	13
<b>2</b>	<b>Common xrd Configuration Directives .....</b>	<b>15</b>
2.1	adminpath .....	15
2.1.1	Administrative Interface .....	16
2.2	allow .....	17
2.3	port .....	19
<b>3</b>	<b>Esoteric xrd Configuration Directives .....</b>	<b>21</b>
3.1	buffers .....	21
3.2	network .....	23
3.3	protocol .....	25
3.4	report .....	27
3.5	sched .....	29
3.6	timeout .....	31
3.7	trace .....	33
<b>4</b>	<b>Common xrootd Configuration Directives .....</b>	<b>35</b>
4.1	export .....	35
4.2	fslib .....	37
4.3	seclib .....	38
<b>5</b>	<b>Esoteric xrootd Configuration Directives .....</b>	<b>39</b>
5.1	async .....	39
5.2	chksum .....	43
5.3	log .....	45
5.4	monitor .....	47
5.5	pidpath .....	51
5.6	prep .....	53
5.7	redirect .....	55
5.8	trace .....	57
<b>6</b>	<b>rootd Configuration Directives .....</b>	<b>59</b>
<b>7</b>	<b>Document Change History .....</b>	<b>61</b>



## 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
<b>xrd</b>	Extended Request Daemon
<b>acc</b>	Access control (i.e., authorization)
<b>cms</b>	Cluster Management Services
<b>ofs</b>	Open File System
<b>oss</b>	Open storage system (i.e., file system implementation)
<b>sec</b>	Security authentication
<b>xrootd</b>	The <b>xrootd</b> protocol implementation.
<b>all</b>	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**, **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 Files Created by xrootd

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

Default File	Changed by	Contents
<stderr>	<b>-l</b> and <b>-n</b> command line options	Informational and error messages
/tmp/.xrootd/admin	<b>-n</b> command line option and the <b>adminpath</b> directive	Local administrative interface.
<cwd>/core	<b>-n</b> command line options	Core file
/tmp/[name/]xrootd.pid	<b>pidpath</b> and <b>-n</b> option	Holds the process id

### 1.3 Starting the xrootd Daemon

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

```
xrootd [ options ] [ path [ path [ . . . ] ] ]
options:  [-c cfn ] [-k {num | sz{k|m|g}}] [-l fn]
          [esoteric]
esoteric: [-b] [-d] [-h] [-n name] [-p {port | any}]
          [-P protocol] [-R user] [-s pfn]
```

#### Function

Start an **xrd**-based daemon implementing the **xrootd** protocol.

#### 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.

**-k** *num* | *sz*{**k**|**m**|**g**}

Keep no more than *num* old log files. If *sz* is specified, the number of log files kept (excluding the current log file) is determined by how much space they use. Hence, kept log files will not exceed *sz* bytes. The *sz* must be suffixed by **k**, **m**, or **g** to indicate **kilobytes**, **megabyte**, or **gigabytes**, respectively.

**-l** *fn* Directs error messages and any trace output to the indicated file, *fn*. By default, messages are directed to standard error.

## Esoteric Options

- b** Runs the program in the background. You should also specify **-l**.
- d** Turns on debugging.
- h** Displays help information.
- 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. 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 *pfm*** Specifies the name of the file that is to hold the process id upon start-up.

## Defaults

```
xrootd -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, 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. This allows you to write an external time-driven script that rotates **xrootd** log files.

### 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. By design, there can only be one logical instance of a manager or server-supervisor pair running on the same machine. The **-n** option allows you to create new logical instances by assigning each instance a different name.
- 5) Since the instance name is used to properly pair **xrootd**'s with **cmsd**'s, specify instance names consistently for **xrootd**'s and **cmsd**'s whether by the **-n** option or using the **name** configuration option.
- 6) The instance name is automatically 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.
- 7) 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. This allows core files to be segregated by instance name.

- 8) 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.
- 9) The **-b** option forces the program into the background. If **-l** is not specified; all output messages are discarded.
- 10) 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.
- 11) 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.
- 12) The **-b**, **-p**, and **-s** command line options are meant to be used by start-up scripts (e.g. **init.d**).
- 13) **Warning**: Command line options, except for **-s**, over-ride corresponding configuration file directives.

### Example

```
xrootd -c /opt/xrootd/xrootd.cf
```

### 1.3.1 Using the StartXRD Script to Start xrootd

The **StartXRD** script may be used to start **xrootd**. The script options and parameters are:

```
StartXRD [-c cfn ] [-D] [-t] [-v] [other]
```

#### Function

Start an **xrd**-based daemon implementing the **xrootd** protocol.

#### Options

- c *fn*** The name of the configuration file. This overrides the configuration file specified in the **StartXRD** configuration file.
- D** Turns on script tracing.
- t** Turns on test mode (i.e., show but do not execute any command).
- v** Turns on verbose output.
- misc* Other **xrootd** command line options to be passed unchanged to **xrootd**.

#### Notes

- 1) The **StartXRD** script uses a configuration file named **StartXRD.cf** to set default values. The next section describes the variables that need to be set in this configuration. The configuration file must be located in the same directory as the **StartXRD** script.
- 2) The **StartCMS** script also uses the **StartXRD.cf** file.
- 3) Use the **StopXRD** script to stop **xrootd**.

#### 1.3.1.1 StartXrd.cf Configuration File

Use the following table to determine which variables should be set to provide appropriate defaults for **StartXRD** as well as **StartCMS**. You will find the **StartXRD.cf** file packaged as **StartXRD.cf.example** to avoid over-writing any existing **StartXRD.cf** file that you may have constructed.

Variable	Contents
XRUSER	The username that is to run <b>xrootd</b> or <b>cmsd</b> . If running as root, an “su” is made to <b>XRUSER</b> . Otherwise, if the current user is not <b>XRUSER</b> ; an error message is issued and the script exits.
XRDBASE	Is the base path for the bin, etc, and lib subdirectories. By default, this is the parent directory of <b>StartXRD.cf</b> .
XRDARCH	Is the architecture directory for the bin and lib directories. By default, if an “arch” or “arch_dbg” subdirectory exists in the <b>XRDBASE/bin</b> directory, this becomes the <b>XRDARCH</b> .
XRDCFG	Is the directory where the <b>xrootd</b> and <b>cmsd</b> configuration file resides. By default this is <b>XRDBASE/etc</b> .
XRDCONFIG	Is the name of the configuration file. By default, it is <b>xrootd.cf</b> in <b>XRDCFG</b> .
XRDHOMEDIR	Is the home directory for <b>xrootd</b> . This is where core files are placed. By default, it is <b>XRDBASE/core</b> .
CMSHOMEDIR	Is the home directory for <b>cmsd</b> . This is where core files are placed. By default, it is <b>XRDBASE/core</b> .
XRLOGDIR	Is the directory for log files. By default, it is <b>XRDBASE/logs</b> .
XRLOGFN	Is the name of the <b>xrootd</b> log file in <b>XRLOGDIR</b> . By default, it is <b>xrdlog</b> .
CMSLOGFN	Is the name of the <b>cmsd</b> log file in <b>XRLOGDIR</b> . By default, it is <b>cmslog</b> .

By default, the following files are either used or created by **xrootd** and **cmsd** when **StartXRD** is used.

File	Construction Method
The <b>cmsd</b> executable	<code>\$XRDBASE/bin/\$XRDARCH/cmsd</code>
The <b>xrootd</b> executable	<code>\$XRDBASE/bin/\$XRDARCH/xrootd</code>
Configuration file for <b>cmsd</b> and <b>xrootd</b>	<code>\$XRDBASE/etc/\$XRDCONFIGFN</code>
Libraries required by <b>cmsd</b> and <b>xrootd</b>	<code>\$XRDBASE /lib/\$XRDARCH<sup>1</sup></code>
The <b>cmsd</b> log file	<code>\$XRLOGDIR/[&lt;name&gt;]/<sup>a</sup>\$CMSLOGFN</code>
The <b>xrootd</b> log file	<code>\$XRLOGDIR/[&lt;name&gt;]/<sup>a</sup>\$XRLOGFN</code>
The <b>cmsd</b> home directory	<code>\$CMSHOMEDIR/core/[&lt;name&gt;]</code>
The <b>xrootd</b> home directory	<code>\$XRDHOMEDIR/core/[&lt;name&gt;]</code>

<sup>1</sup> This path is added to LD\_LIBRARY\_PATH environmental variable.

<sup>a</sup> <name> is the **-n** argument (i.e., instance name) and is automatically added by **xrootd** if **-n** is specified.

<sup>a</sup> <name> is the **-n** argument (i.e., instance name) and is automatically added by **xrootd** if **-n** is specified.

### 1.3.2 Using the StopXRD Script to Stop xrootd

The **StopXRD** script may be used to stop **xrootd**. The script options and parameters are:

```
StopXRD [-c] [-D]
```

#### Function

Stop an **xrd**-based daemon implementing the **xrootd** protocol.

#### Options

- c** Allows the script to end normally even when **xrootd** was not found to be executing.
- D** Turns on script tracing.

#### Notes

- 1) The **StopXRD** script attempts to kill any process running “**xrootd**”.
- 2) It does *not* read the **StartXRD.cf** configuration file.



## 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 its 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 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 it 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.

#### **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 **kilo-**, **mega-**, or **giga-**bytes; respectively. The default is to use up to 12.5% (one-eighth) of the configured memory of the machine.

*rint* The interval between buffer pool readjustments. Specify a number, optionally suffixed by **m** or **minutes**, **s** for **seconds** (the default), or **h** for **hours**. 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 [wan] [buffsz blen[k | m | g]] [keepalive]
           [ [no] dnr ]
```

### Function

Specify network parameters.

### Parameters

**wan** Indicates that the subsequent specifications apply to the special **Wide Area Network** port.

### **buffsz** *blen*

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

### **keepalive**

Uses the operating system's keep-alive mechanism to determine whether or not a client is still connected to the daemon. The default is to use idle socket time-outs (see the `timeout` directive).

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

**nodnr** Does not use Domain Name Resolution to convert IP addresses to host name for connecting clients. Client IP addresses are displayed in various messages.

### Defaults

The operating system's socket buffer size is used but its **keepalive** mechanism is not used. Domain Name Resolution is turned on (i.e., **dnr**).

### Notes

- 1) For systems that support TCP buffer auto-tuning as a manual option, specify a *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 than 64k.
- 3) Normally, the best performance is obtained by using TCP buffer auto-tuning.
- 4) By default, Domain Name Resolution is turned on (i.e., **dnr**). This is to allow for host-based authorization using the Access Control Component. If you are not using host-based authorization or are only using ip-addresses, you can avoid DNS latencies so as to maximize the client connection rate by turning **DNR** off (i.e., **nodnr**). Domain Resolution may be forced on if you specify an **allow** directive using a host name or host name fragment. Resolution is still optimized by caching the results for future use.
- 5) Generally, the daemon's internal timeout mechanism is sufficient to discover unconnected clients.
- 6) The **wan** port is created only if there is a protocol that can handle wan connections. See the protocol directive.

### Example

```
xrd.network buffsz 128k keepalive nodnr
xrd.network wan buffsz 512k
```

### 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 that 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** or **minutes**, **s** for **seconds** (the default), or **h** for **hours**. 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:

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

##### **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** or **minutes**, **s** for **seconds** (the default), or **h** for **hours**. 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 kilobytes or **m** for megabytes. 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 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
```

### 3.7 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:

<b>all</b>	selects all possible trace levels
<b>conn</b>	traces connection activity
<b>debug</b>	traces internal activities for debugging purposes
<b>mem</b>	traces memory management functions
<b>net</b>	traces network management functions
<b>none</b>	traces nothing
<b>off</b>	a synonym for <b>NONE</b>
<b>poll</b>	traces I/O interrupt polling activities
<b>protocol</b>	traces protocol activity (see the notes)
<b>sched</b>	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.

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

#### **no**lock

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 **no**lock option to disable the default.
- 4) The **[no]lock** option *must* appear *before* any **oss** options.

**Example**

```
xrootd.export /store
```

## 4.2 **fslib**

```
xrootd.fslib [?] path
```

### Function

Specify the location of the file system interface layer.

### Parameters

? Checks that the **fslib** version equals the version of the **xrootd** executable. A warning message is printed if the versions do not match.

*path* The absolute 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

A built-in minimal native **ofs** implementation is used.

### Notes

- 1) The **ofs** interface allows you to provide an arbitrary file system implementation (e.g., Mass Storage, Dynamic Load Balancing, etc).
- 2) Refer to the “**ofs** Developer’s Reference” on how to write an **ofs** interface.

### Example

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

### 4.3 seclib

```
xrootd.seclib path
```

#### Function

Specify the location of the security interface layer.

#### Parameters

*path* The absolute path to the shared library that contains an implementation of the Security (sec) interface that **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.hh** 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]] [minfsz 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 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 **kilo-**, **mega-**, or **giga-**bytes; respectively. The default is one half of the initial *segsz*.

**minfsz** *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 **kilo-**, **mega-**, or **giga-**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 **kilo-**, **mega-**, or **giga-**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 [max num] digest [path [args]]
```

### Function

Specify how file check sums are computed.

### Parameters

*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.

*digest* The name of the checksum digest (e.g., md5) used for the check summing.

*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 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.
  - b. 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.
  - c. 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.

- d. The result of one of the two previous steps is reported 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 passed as the last argument and is the only argument if no *args* have been specified.
- 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 administrator's interface allows you to list and cancel checksum jobs. This applies to external as well as internal computation of the checksum.
- 12) 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 checksum** command. See the File Residency Manager reference.

### Example

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

## 5.3 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:

<b>all</b>	logs all possible events, the default
<b>disc</b>	disconnect events
<b>login</b>	login events

### Defaults

```
xrootd.log all
```

### Notes

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

### Example

```
xrootd.log all -login
```



## 5.4 monitor

```

xrootd.monitor [ options ] dest [ dest ]

options: [ all ] [ auth ] [ flush [ io ] intvl[m|s|h]]

           [ ident sec ] [ mbuff size[k] ] [ rbuff size[k]]

           [ rnums cnt ] [ window intvl[m|s|h]]

dest:    dest events host:port

events:  [ files ] [ 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.

### **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 **kilo**-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 **kilo**-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** *intol*

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).
- io** I/O request monitoring (read and write requests plus files).
- iovs** 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. The data may then be used to determine access patterns or used as an I/O trace for simulation studies.
- 2) The **flush** parameter does *not* apply to monitor streams that include **io** event data. 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.
- 3) 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**, and **user**).
- 4) 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.
- 5) 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.
- 6) Clients cannot enable monitoring that has not been enabled by the **monitor** directive.
- 7) Specifying a small datagram buffer size (less than 4k) 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 (each connection allocates a buffer) and latency in reporting events.
- 8) Approximately 61 requests can fit into a 1K **mbuff**.
- 9) On average, 64 to 128 redirection events can fit into a 32K **rbuff**.
- 10) Increasing the number of redirection monitoring streams (**rnums**) reduces the bottlenecks in the monitoring path.
- 11) 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.
- 12) Refer to the “Scalla Monitoring” reference for a detailed explanation on the datagram format used by the monitoring subsystem.

## Example

```
xrootd.monitor all window 5m dest io datacoll:5050
```



## 5.5 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
```



## 5.6 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
```

## 5.7 redirect

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

### Function

Enable meta-data command forwarding.

### Parameters

*host:port*

The name of the *host* and *port* number where clients are to be redirected based on the subsequent parameters.

*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:

<b>all</b>	redirect all possible operations
<b>chmod</b>	redirect change mode requests
<b>chksum</b>	redirect checksum requests
<b>dirlist</b>	redirect directory content listing requests
<b>locate</b>	redirect path location requests
<b>mkdir</b>	redirect create directory requests
<b>mv</b>	redirect rename requests
<b>prepare</b>	redirect prepare requests that <i>do not need</i> file staging
<b>prepstage</b>	redirect prepare requests that <i>may need</i> file staging
<b>rm</b>	redirect file removal requests
<b>rmdir</b>	redirect directory removal requests
<b>stat</b>	redirect file attribute requests
<b>trunc</b>	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., EONONENT) 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

- 1) 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.
- 2) 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.
- 3) The `redirect` path prefixes are always matched from most- to least-specific prefix (i.e., longest to shortest).

### Example

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

## 5.8 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 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:

<b>all</b>	selects all possible trace levels
<b>debug</b>	traces internal activities for debugging purposes
<b>emsg</b>	traces errors sent back to the client
<b>fs</b>	traces file system requests
<b>login</b>	traces login and authentication steps
<b>mem</b>	traces memory management functions
<b>none</b>	traces nothing
<b>off</b>	a synonym for <b>none</b>
<b>stall</b>	traces client deferrals due to resource limitations
<b>redirect</b>	traces client redirections to other servers
<b>request</b>	traces client request information
<b>response</b>	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 rootd Configuration Directives

The **rootd** protocol is implemented as an interface to the standard **rootd** daemon. The interface code is part of the standard **xrootd** distribution and is packaged in the **librootd.so** shared library. The interface needs to know how to run the root daemon should a connection be made by a client that does not speak **xrootd** protocol. The command to run is specified as part of the **xrd** protocol directive. For instance,

```
xrd.protocol rootd /usr/lib/librootd.so \  
                /usr/local/bin/rootd -i
```

The **-i** option tells **rootd** it is being started as if **inetd** intercepted the connection (i.e., the incoming socket is connected to standard in and standard out). In many ways, the **rootd** interface protocol provides an **inetd**-like interface to an arbitrary command, as long as the connection *appears* to be talking **rootd** protocol.

Refer to the “ROOT User’s Guide” for complete information on how to set up a root daemon. When configuring to run **rootd** as a protocol you *must*:

- have **librootd.so** installed in an accessible place. The location is not material.
- specify the location of the **librootd.so** in the **xrd** configuration file.
- have the **rootd** executable installed in an accessible place. The location is not material.
- specify the location of the **rootd** executable as a parameter to the **rootd** protocol specification.
- Start **rootd** with the **-i** option (run in **inetd** mode).



## 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 minfsz** 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.