XRootD

Open File System & Open Storage System Configuration Reference

8 December 2021
Release 5.4.0 and above

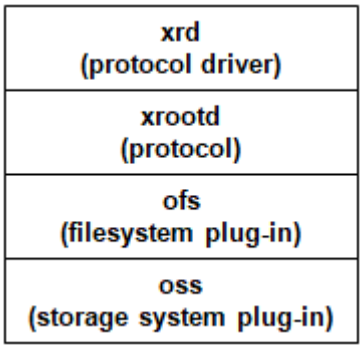
Andrew Hanushevsky

SLAC
NATIONAL ACCELERATOR LABORATORY

1 Introduction

This document describes Scalla configuration directives for the **O**pen **F**ile **S**ystem and the **O**pen **S**torage **S**ystem components.

Configuration directives for each component come from a configuration file. The **xrootd** structure requires that all components read their directives from the same configuration file. This is the configuration file specified when **xrootd** was started (see the **–c** option **xrootd** option). This is possible because each component is identified by a unique 3-letter prefix. This allows a common configuration file to be used for the whole system.



The particular components that need to be configured are the file system plug-in (**ofs**) and the storage system plug-in (**oss**). The relationship between **xrootd** and the plug-ins is shown on the left. The protocol driver (**xrd**) runs the **xroot** protocol which, in turn, utilizes the file system plug-in that, itself, relies on the storage system plug-in. Collectively, this is called **xrootd**, the executable that encapsulates all of the components.

The prefixes documented in this manual are listed in the following table. The **all** prefix is used in instances where a directive applies to more than one component. *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, #).

Prefix	Component
`ofs`	**O**pen **F**ile **S**ystem coordinating **acc**, **cms**, & **oss** components
`oss`	**O**pen **S**torage **S**ystem (i.e., storage system implementation)
`all`	Applies the directive to the above components.

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.

2 Common ofs Configuration Directives

2.1 authorize

```
ofs.authorize
```

Function

Enable the access control, **acc**, component.

Parameters

None.

Defaults

The access control component is normally disabled.

Notes

1) When the **acc** component is disabled, all uses have the ***same*** access rights to directories and files as is afforded to the user running **xrootd**. Because of this, **xrootd** should ***never*** be executed with superuser (i.e., root) privileges.
2) Refer to the "**xrootd** Authentication and Authorization Reference" on how to configure the access control component.

Example

```
ofs.authorize
```

3 Esoteric ofs Configuration Directives

3.1 authlib

```
ofs.authlib [++] path [parms]
```

Function

Specify the location of the file system authorization interface layer.

Parameters

++ The specified plug-in should stack on top of the existing plug-in or default. A stacked plug-in cannot be overridden by a subsequent directive.

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

parms Optional parameters to be passed to the authorization object creation function.

Defaults

A built-in minimal implementation is enabled for use when the **authorize** directive is specified.

Notes

1) The authorization interface is defined in the **XrdAccAuthorize.hh** include file. Refer to this file on how to create a custom authorization algorithm.
2) You must specify the **authorize** directive in order to enable the authorization interface.

Example

```
ofs.authlib /opt/xrootd/lib/libAuth.so
```

3.2 chkpnt

```
ofs.chkpnt [disable | enable] [cprerr {makero | stopio}]
           [maxsz size[k | m | g]] [path path]
```

Function

Specify checkpoint parameters.

Parameters

disable

Disables the processing of checkpoint requests. This is the default when *path* is rooted in **/tmp**.

enable

Enables the processing of checkpoint requests. This is the default unless *path* is rooted in **/tmp**; in which case **disable** becomes the default. However, when **enable** is actually specified, *path* is allowed to be rooted in **/tmp**.

cprerr Specified the disposition of a file that could not be restored from a checkpoint. The options are:

makero - makes the file read/only.

stopio - makes the file inaccessible either for reading or writing. This is the default.

maxsz *size*[**k** | **m** | **g**]

Specifies the maximum number of modified bytes a checkpoint file may hold. Specify for *size* a value no less than **10m**. The value may be suffixed by **k**, **m**, or **g** to scale *size* by 2^{10}, 2^{20}, or 2^{30}, respectively. The default is **10m**.

path *path*

Specifies the absolute path of the directory where checkpoint files are to be placed. The *path* is suffixed by the instance name (**-n** command option), if any, and "**chkpnt**" as additional directory components. The default *path* is "*adminpath*[/*name*]/**chkpnt**". The path is created if it does not exist. See the notes on how to choose an appropriate path.

Defaults

ofs.chkpnt enable cpprerr stopio maxsz 10m
path *adminpath*[**/***name*]**/chkpnt**

Notes

1) You should choose a checkpoint directory path that is robust and has sufficient space to hold the expected number of checkpoint files of the given or default size. Make sure that the default is appropriate.
2) Checkpoint directories rooted in **/tmp** are not appropriate and this is why checkpoint processing is disabled by default if this is the case unless you explicitly specify **enable**.
3) Active checkpoint files created in this directory are suffixed with "**.ckp**". Checkpoint files that could not be used to restore a file because of data corruption have a suffix of "**.ckperr**".
4) The system automatically restores files with any active checkpoints during initialization. If a checkpoint file cannot be used to restore a file during normal processing or during initialization, the log file should contain the reasons for the failure. The active checkpoint is then renamed and the file's accessibility is changed as indicated by the **cprerr** option.
5) Files that could not be restored should be replaced with backup copies. You can always determine which file a checkpoint targets by either
   a. **getfattr -d** *ckpfile* (displays [...]**xrdckp_srclfn=**"*target_lfn***"**) or
   b. **strings** *dkpfile* | **grep "file://"** (displays " **file:**///*taget_lfn*").
6) Use the **getfattr** command only if the file system supports extended file attributes. Otherwise, use the **string-grep** method. The latter may not yield the actual target logical file name if the checkpoint file is severely corrupted.
7) If you decide to change the location of the checkpoint directory you must make sure there are no active checkpoint files in the old directory. If there are active checkpoints then you must copy them to the new directory after shutting down the server.

Example

ofs.chkpnt cprerr makero maxsz 20m path /raid1/ckpdata

3.3 ckslib

```
ofs.ckslib {digest | * | = | ++} path [parms]
```

Function

Specify a plug-in for checksum management or calculation.

Parameters

digest A 1- to 15-character name of a checksum digest identifying the checksum algorithm to be loaded.

***** loads a the checksum manager to be used at the storage endpoint (i.e. the server that holds the file).

= loads a the checksum manager to be used at the manager node (i.e. the redirector).

++ The specified plug-in should stack on top of the existing checksum manager plug-in or default. A stacked plug-in cannot be overridden by a subsequent directive.

path The absolute path to the shared library that contains an implementation of the checksum management interface (when digest is an asterisk) that **ofs** uses to compute checksums or an implementation of a checksum calculation with an interface used by the checksum manager.

parms Optional parameters to be passed to the checksum object creation function.

Defaults

The **adler32**, **crc32**, **crc32c**, and **md5** digests are natively supported. A built-in manager is used to manage these checksums.

Notes

1) The checksum calculation interface is defined in the **XrdCksCalc.hh** include file. Refer to this file on how to create a custom digest.
2) The checksum management interface is defined in the **XrdCks.hh** include file. Refer to this file on how to create a custom checksum manager.

3) Four checksums are pre-defined: **adler32**, **crc32**, **crc32c**, and **md5**. You may supply a custom implementation for any of these or add one additional custom digest.
4) The **crc32c** checksum uses hardware assist should the platform support it (e.g. AMD and Intel).
5) You may wish to implement platform-specific digests to improve checksum performance.
6) Normally, checksums are handled by the server that holds the file in question. If a custom checksum manager can provide the same facilities running on a manager node, then use an equal sign before *path*. When loaded on a manager node, redirection is suppressed and requests for checksum services are performed on the manager's node.
7) Use the **pss.ckslib** directive to specify a checksum plug-in for a proxy.

Example

```
ofs.ckslib sha256 /opt/xrootd/lib/libCksSHA256.so
```

3.4 cksrdsz

```
ofs.cksrdsz num[k | m | g]
```

Function

Specify the checksum read siz.

Parameters

num The maximum number of bytes to be read at a time when computing a file's checksum. The minimum is **64k** and the maximum is **1g**. Intermediate quantities are forced to be multiples of 64k. The quantity may be suffixed by **k**, **m**, or **g** to scale *num* by 2^{10}, 2^{20}, or 2^{30}, respectively.

Defaults

ofs.cksrdsz 64m

Notes.

1) The maximum amount of memory consumed when calculating checksums is determined by the **max** parameter of the **xrootd.chksum** directive multiplied by the value of the **ofs.cksrdsz** parameter.

Example

ofs.cksrdsz 32m

3.5 cmslib

```
ofs.cmslib path [parms]
```

Function

Specify the location of the cluster management client interface layer.

Parameters

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

parms Optional parameters to be passed to the cluster management client object during configuration.

Defaults

A full-featured built-in implementation is enabled for use by the **ofs** layer when the **all.role** directive is specified.

Notes

1) The cluster manager client interface is defined in the **XrdCmsClient.hh** include file. Refer to this file on how to create a custom cluster management client implementation.

Example

```
ofs.cmslib /opt/xrootd/lib/libmyCms.so
```

3.6 ctllib

```
ofs.ctllib [++] path [parms]
```

Function

Specify the location of the file system dependent control request implementation (i.e. file system **FSctl()** and file directed **fctl()**).

Parameters

++ The specified plug-in should stack on top of the existing plug-in or default. A stacked plug-in cannot be overridden by a subsequent directive.

path The path to the shared library that contains an implementation of the **FSctl** operation.

parms Optional parameters to be passed to the authorization object creation function.

Defaults

There is no default and normally when requested without an implementation a "not supported" error is returned.

Notes

1) The **FSctk** plug-in interface is defined in the **XrdOfsFSctl_PI.hh** include file. Refer to this file on how to create a custom **FSctl()** and **fctl()** implementations.
2) The plug-in is invoked in response to the **kXR_query** request for **kXR_Qopaque**, **kXR_Qopaquf**, and **kXR_Qopaqug** query codes.

Example

```
ofs.ctllib /opt/xrootd/lib/libFSCtl.so
```

3.7 dirlist

```
ofs.dirlist { local | remote }
```

Function

Specify how directory listings are to be performed. This is the default.

Parameters

local If the server is clustered then it ask the cluster manager for the location of the directory to be listed. Otherwise, it lists the directory that exists on the server.

remote

The path to the shared library that contains an implementation of the **FSctl** operation.

Defaults

```
ofs.dirlist local
```

Notes

1) The **ofs.dirlist** directive is intended to support Open Storage System plug-ins that perform directory listings on the server being asked and Cluster Management plug-ins that can list remote directories.
2) In practically all cases, the default provides the reasonable action.

Example

```
ofs.dirlist remote
```

3.8 forward

```
ofs.forward [1way | 2way | 3way host:port] opers

opers:    [-]foper [ opers ]

foper:    all | chmod | mkdir | mv | remove | rm | rmdir |
          trunc
```

Function

Enable meta-data command forwarding.

Parameters

1way The request is forwarded with no expectation of a response from the **cmsd**. This is the default forwarding mode.

2way The request is forwarded with an expectation of a response from the **cmsd**. The is asked to wait until the **cmsd** provides a response to the request.

3way The request is forwarded with in 1way mode (see above). Afterwards, the client is redirected to *host*:*port* and typically re-issues the request there.

host is the name or ip address of the host to receive the redirected client request.

port is the port number the redirected client is to use.

foper Specifies which metadata operations are to be forwarded. One ore 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	all possible operations	**remove**	same as **rm** and **rmdir**
chmod	change mode	**rm**	file removal
mkdir	create directory	**rmdir**	directory removal
mv	rename requests	**trunc**	truncate by filename

Defaults

ofs.forward -all

Notes

1) Request forwarding is only 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, in a clustered environment there is a choice of whether to perform the operation on the host that has the target object or every host that may have the target object. When the **forward** 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 forwarded, the operation is automatically sent to all hosts that can potentially have the object in question and the client is not redirected, unless **3way** forwarding is in effect.
3) Forward and non-forward modes of operation have distinctly different semantics. However, neither mode takes into account that the list of potential servers is dynamic. Since ***it is not possible to define a semantically consistent result***; neither mode may be suitable.
4) By default, when a request is forwarded, the client is not notified whether or not it succeeded because the notion of success is undefined. However, the request is executed on every currently active host that could have possibly executed the request successfully.
5) If you provide a custom cluster manager interface using the clustering **xmilib** directive, forwarded requests are processed by the provided implementation. Client interactions and semantic results are then defined by the custom implementation.

Example

```
ofs.forward mv rm
```

3.9 maxdelay

```
ofs.maxdeley sec
```

Function

Specify how long a client may be delayed each time.

Parameters

sec The maximum number of seconds a client may be when a delay is requested by any other component.

Defaults

```
ofs.maxdelay 60
```

Notes

1) Each component coordinated by **ofs** may request that a client be delayed a specific amount of time to allow for request processing. The **maxdelay** directive tells **ofs** the maximum amount of delay that may be imposed each time it is requested. If the delay requested by a component exceeds the **maxdelay** amount, **ofs** uses the **maxdelay** amount.
2) The default value is usually sufficient except for unusual cases. Reducing the value may cause more client activity that would otherwise be necessary. Increasing the value may introduce unacceptable latencies.

Example

```
ofs.maxdelay 45
```

3.10 notify

```
ofs.notify events [msgs nums [numl]] [| | >]command

events: [-]event [ [-]event ] [• • •]

event:  all | chmod | close | closer | closew | create

        fwrite | mkdir | open | openr | openw | mv |

        rm | rmdir | trunc
```

Function

Enable event forwarding.

Parameters

events Specifies which operations are to send an event notification to *command*. 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	sends notifications for all the following operations
chmod	access mode change
close	a file is physically closed
closer	a file opened in read-only mode is physically closed
closew	a file opened in read-write mode is physically closed
create	a file opened for possible creation
fwrite	the first write to a file after it was opened in read-write mode
mkdir	a directory create operation
mv	a rename operation
open	a file is physically opened
openr	a file is physically opened in read-only mode
openw	a file is physically opened in read-write mode
rm	file removal
rmdir	directory removal
trunc	file truncation via filename

nums The maximum number of short message objects that may be kept on hand for future use. The default is 90. See the usage notes on how this value limits the maximum number of queued messages.

numl The maximum number of large message objects that may be kept on hand for future use. The default is 10. See the usage notes on how this value limits the maximum number of queued messages.

command

The command, or FIFO, that is the target for event notifications. If *command* is preceded by "**>**" then a FIFO named *command* is created and notifications are written to the FIFO. No program is started. If *command* is preceded by "**|**", the default, notifications are sent to *command* via STDIN after *command* is started. See the next section on the format of event notification messages.

Defaults

Event notification is normally disabled. Should is enabled, "**`msgs 90 10`**" is the default.

Notes

1) If *command* is not preceded by "**>**" then *command* is started at initialization time and is expected to consume input on STDIN.
2) The quantity *nums*+*numl* sets the limit on the number of queued messages to command. When the limit is exceeded, messages are discarded and a warning appears in the log when this occurs.

Example

```
ofs.notify closew |/usr/bin/xevent
```

3.11 notifymsg

```
ofs.notifymsg event msgline

event: chmod | closer | closew | create | fwrite | mkdir |
       openr | openw | mv | rm | rmdir | trunc

msgline: [text] [var] [msgline]

var:    $CGI | $CGI1 | $CGI2 | $FMODE | $FSIZE | $LFN |
        $LFN1 | $LFN2 | $TID
```

Function

Specify the message to send to a destination when a monitored event occurs.

Parameters

event Specifies which operation, when enabled, is to generate a msgline. Specify one of the following:

chmod	**closew**	**fwrite**	**mv**	**openw**	**rmdir**
closer	**create**	**mkdir**	**openr**	**rm**	**trunc**

text Arbitrary text.

var A variable whose value is determined by the current request setting. The following variables may be specified:

- **$CGI** all of the opaque information specified after the question mark in the file path (same as **$CGI1**).
- **$CGI1** all of the opaque information specified after the question mark in the first file path of two possible paths (same as **$CGI**).
- **$CGI2** all of the opaque information specified after the question mark in the second file path of two possible paths.
- **$FMODE** the file access mode in octal.
- **$FSIZE** the file size in bytes.
- **$LFN** the logical file name associated with the event (same as **$LFN1**).
- **$LFN1** the *first* logical file name of two possible names (same as **$LFN**).
- **$LFN2** the *second* logical file name of two possible names.a rename operation as modified by **localroot** or the **namelib** function

$TID the client's trace identity as *username*.*pid*:*fd*@*host* where:

username	is the unauthenticated name of the client's user.
pid	is the client's process id.
fd	is the server's socket file descriptor number.
host	is the name of the originating host.

Defaults

The default message is described in the following section.

Notes

1) You may define notification messages with listing them, thus enabling them, via the **notify** directive.
2) Messages are automatically ended with a new-line character ('\n').
3) Variables must begin with a $ (dollar sign) and end with a non-alpha-numeric character.
4) To include a dollar sign into the message, escape it with a back slash ("\").
5) A backslash escape is only recognized when followed by a dollar sign.
6) Important! The **notifymsg** *msgline* is ***not*** subject to general **set** variable substitution.
7) Except for **$CGI** and its variants, the implicit value of a variable that has not been set is the variable name itself, including the dollar sign.
8) For **$CGI**, if no opaque information is found, the variable is substituted with the null string.
9) Not all variables are meaningful for all events. The following table lists which variables have values.

Event	Meaningful Variables
chmod	$CGI, $CGI1, $FMODE, $LFN, $LFN1, $TID
closer	$CGI, $CGI1, $LFN, $LFN1, $TID
closew	$CGI, $CGI1, $FSIZE, $LFN, $LFN1, $TID
create	$CGI, $CGI1, $FMODE, $LFN, $LFN1, $TID
fwrite	$LFN, $LFN1, $TID
mkdir	$CGI, $CGI1, $FMODE, $LFN, $LFN1, $TID
mv	$CGI, $CGI1, $CGI2,, $LFN, $LFN1, $LFN2, $TID
openr	$CGI, $CGI1, $LFN, $LFN1, $TID
openw	$CGI, $CGI1, $LFN, $LFN1, $TID
rm	$CGI, $CGI1, $LFN, $LFN1, $TID
rmdir	$CGI, $CGI1, $LFN, $LFN1, $TID
trunc	$CGI, $CGI1, $FSIZE, $LFN, $LFN1, $TID

Example

```
ofs.notifymsg chmod $NAME $LFN $MODE
```

3.11.1 Default Event Notification Messages

Event notification messages are always sent as new-line (\n) terminated ASCII text. The first token identifies the client that generated the event and the second token is the name of the operation that generated the event. Subsequent tokens must be interpreted in the context of the operation. The event messages are:

```
$TID chmod $FMODE $LFN1
$TID closer $LFN1
$TID closew $LFN1
$TID create $FMODE $LFN1
$TID fwrite $LFN1
$TID mkdir $FMODE $LFN1
$TID mv $LFN1 $LFN2
$TID openr $LFN1
$TID openw $LFN1
$TID rm $LFN1
$TID rmdir $LFN1
```

3.12 osslib

```
ofs.osslib [++ | [+cksio] [+xattr]] path [parms]
```

Function

Specify the location of the storage system interface layer.

Parameters

++ The specified plug-in should stack on top of the existing plug-in or default. A stacked plug-in cannot be overridden by a subsequent directive.

+cksio the specified **osslib** is to also handle all I/O related to computing checksums. Normally, native file system calls are used and this remains the default for backward compatibility. However, when the default **Oss** plug-in is used, **+cksio** is the default.

+xattr Specifies that the extended attribute plug-in should be loaded from the library identified by *path*. The plug-in is also passed *parms*, if any.

path The absolute path to the shared library that contains an implementation of the storage system interface that **ofs** is to use for storage access for file system specific operations (e.g., open, close, read, write, rename, etc). Do *not* include the version number (i.e. ***–n*** appearing before '**.so**') in the specification.

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

Defaults

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

Notes.

1) The **+xattr** option provides a convenient way to package an **oss** and extended attribute plug-ins together in a single shared library. If the plug-ins reside in separate libraries or if each needs separate parameters, use the **ofs.xattrlib** directive as well.
2) For developers: when **+cksio** is in effect, the checksum manager receives logical file names as opposed to physical file names.

Example

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

3.13 persist

```
ofs.persist  [mode] [opts]

mode:     auto | manual | off

opts:     hold hval[s|h|m] | logdir path | sync rnum | opts
```

Function

Set persist-on-successful-close (**POSC**) parameters.

Parameters

auto **POSC** processing applies to all create and create/truncate open requests.

manual

POSC processing applies only when explicitly requested by the client. This is the default.

off Disables **POSC** processing.

hval is the amount of time an incomplete file may remain in the system waiting for a successful close request before it is deleted. The value may be suffixed by **s**, **h**, or **m** (the case is immaterial) to indicate that seconds (the default), minutes, or hours are being specified, respectively.

path is the directory where the **POSC** log file is to be placed. The default is based on the **all.adminpath** directive or its default value when not specified. The system automatically attempts to create any missing path components of *path*.

rnum the number of outstanding open requests allowed before the open request log is committed to disk. The default is 0 which commits each open request to disk. Specify a number between 0 and 32767, inclusive.

Defaults

ofs.persist manual hold 10m logdir *adminpath*[**/**name]**/.ofs sync 0**

Notes

1) When a file is open in create mode or create-truncate mode with **POSC** processing enabled; **xrootd** requires that the file be explicitly closed for it to persist (i.e., not be erased). This prevents partially copied files from masquerading as fully complete files. If the client disconnects or the server fails before the file is closed, the file is placed in a pending state for *hval* time and the system waits for a file re-open and a successful close.
2) When a file has been placed in a pending state, it may only be re-opened in read-write or write mode. Clients opening the file in read-mode are delayed until the file is either successfully closed or removed.
3) To minimize the risk of incorrect completion, the system limits which clients can re-open a pending file.
    a. Normally there can only be a single writer with no readers. This is the first line of defense. However, since the **all.export NOLOCK** option bypasses this defense it is incompatible with **POSC** processing unless an external locking mechanism of comparable strength is used.
    b. If the file is re-opened without specifying the **POSC** option (see below) then only the client who has current create ownership is allowed to open the file. Create ownership is first assigned when a file is opened either with the **kXR_new** or **kXR_delete** option (i.e., create or create/trunc). The owner's identity is based on the client's *loginid*, process ID (i.e., **pid**), and host name. Hence, only the client that originally created the file may continue to write to the file.
    c. If the file is re-opened with the **POSC** option, then create ownership is transferred to the client re-opening the file.
4) **POSC** processing may be requested in one of two ways:
    a. Using the **kXR_posc** option on the **kXR_open** request, or
    b. Specifying "**posc.ofs=y**" in the **CGI** part of the file name (e.g., file names of */path/fname***?posc.ofs=y**").
5) When a **POSC**-enabled file encounters a file system related error during a write operation, its name is immediately deleted from the file system. The space is released when the client either closes the file or disconnects.
6) When obtaining stat information for a **POSC**-type file not yet successfully closed, the **kXR_poscpend** flag will be set in the flag area.
7) The **sync** parameter allows you to reduce disk load in order to support more **POSC** open requests per second. Be aware that should the machine fail before the **POSC** log is committed to disk, files nor reflected in the log will not be removed upon restart.

Example

```
ofs.persist auto logdir /var/run/xrootd/posc
```

3.14 preplib

```
ofs.preplib [++ | [+noauth]] path [parms]
```

Function

Specify the location of the file preparation plug-in.

Parameters

++ The specified plug-in should stack on top of the existing plug-in or default. A stacked plug-in cannot be overridden by a subsequent directive.

+noauth

Does not apply authorization to the list of files to be prepared.

path The absolute path to the shared library that contains an implementation of the prepare interface that **ofs** is to use for prepare requests.

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

Defaults

A basic built-in implementation is enabled for use by the **ofs** layer.

Notes

1) The prepare plug-in interface is defined in the **XrdOfsPrepare.hh** include file. Refer to this file on how to create a custom file preparation implementation.
2) Normally, all file paths supplied by the client must be readable by the client. If you wish to perform custom authorization, specify **+noauth** to disable the authorization check.

Example

```
ofs.preplib /opt/xrootd/lib/libmyprep.so
```

3.14.1 Generic Prepare Plug-in

```
ofs.preplib [+noauth] dpath/libXrdOfsPrepGPI.so [parms]

parms:   -admit reqlist [-cgi] [-debug] [-maxfiles nf]

         [-maxquery nq] [-maxreq nr] [-pfn] -run pgm

reqlist: req[,reqlist]

req:     all | cancel | evict | prep | query | stage
```

Function

Specify the location of the generic prepare plug-in and its parameters.

Parameters

dpath The directory path where the plug-in shared library resides.

parms The plug-in parameters. If not specified with the directive, the parameters may be specified in the configuration file using the **prepgpi.parms** directive. See the examples for details.

-admit

The comma separated list of requests the plug-in wishes to handle. Unspecified requests are handled using defaults provided by the plug-in driver (see the notes for more details). The requests that can be admitted are:

all all possible requests (i.e. those below).
cancel requests to cancel previously issued evict or stage requests.
evict requests to remove stage-able files (see usage notes).
prep notification of future access.
query requests to display the state of a previous stage request.
stage requests to bring offline files online for access (e.g. tape).

-cgi Includes the **cgi** information, if present, with the target file name.

-debug

Displays extensive debugging information and should only be used when actually debugging a particular problem.

-maxfiles *nf*

The maximum number of files allowed in a single request. Specify for *nf* a value between 1 and 1024, inclusive. The default is 48.

-maxquery *nq*

The maximum number of concurrent query requests allowed. Requests beyond this value are queued for execution. Specify for *qr* a value between 1 and 64, inclusive. The default is 8.

-maxreq *nr*

The maximum number of concurrent requests allowed. Requests beyond this value are queued for execution. Specify for *nr* a value between 1 and 64, inclusive. The default is 4.

-pfn passes physical file names to the executable. By default, logical file names are passed to the executable.

-run *pgm*

specifies the executable, *pgm*, to launch to process the request. The name of the executable should be an absolute path but this is not enforced.

Defaults

```
-maxfile 48 –maxqueries 8 –maxreq 4
```

Notes

1) Each request has a built-in handling default, as follow:
   a. **cancel**: if the target request is still queued it is removed from the queue. Otherwise, the client is told that the request cannot be cancelled.
   b. **evict**: the client is told the request is not supported.
   c. **prep**: the request is accepted but otherwise ignored.
   d. **query**: the client is told whether or not the target request is queued.
   e. **stage**: the client is told the request is not supported.
2) **Evict** requests are used to indicate that a file does not need to be online. That is, if it were previously staged from an external source it can be removed. If it has not yet been copied to an external source it should be removed after the file has been copied. The actual implementation should adhere to this notion.

3) Query and non-query requests have separate concurrent limits. The **maxquery** parameter specified the maximum number of concurrent queries while the **maxreq** parameter specifies the maximum number of concurrent non-query requests.
4) The generic prepare plug-in is not stackable and must be the last one in the chain of stacked plug-ins.
5) Requests are queued in the order they arrive. Be aware that query and non-query requests have separate queues.
6) A query requests can only remain queued for a maximum of 33 seconds. If it cannot be launched within this time, the client is told the request timed out.
7) All requests accept a list of target files. For instance, files specified for query only request information for the specified files. Whether or not such specificity is supported is implementation dependent.
8) The **xrdprep** command provides a convenient command line interface to the prepare process.

Example

The following two examples are equivalent.

Example 1:

```
ofs.preplib /opt/xrootd/lib/libXrdOfsPrepGPI.so\
            -admit query,stage –maxfiles 256\
            –run /opt/xrootd/preppgm
```

Example 2:

```
ofs.preplib /opt/xrootd/lib/libXrdOfsPrepGPI.so

prepgpi.parms -admit query,stage –maxfiles 256
prepgpi.parms –run /opt/xrootd/preppgm
```

3.14.2 Generic Prepare Program Execution

The following arguments and environmental variable are passed to the launched prepare executable. Note that the number of arguments is variable but options always ends with a '--' and is followed by parameters.

```
argv: pgm [-a] [-C] [-n {fin|rdy}] [-p prty] [-w] -- parms

parms:   handle req [path [path […]]

req:     cancel | evict | prep | query | stage

         ---------------------------------------

Envars:  XRDPREP_COLOC XRDPREP_NOTIFY XRDPREP_TID
```

Arguments

pgm The name of the executable.

-a The access time of the file being prepared should be set to the current time. This is only relevant for **prep** and **stage** requests.

-C The file(s) should be physically co-located with the file named in the **XRDPREP_COLOC** environmental variable. This is only relevant for **prep** and **stage** requests.

-n {**fin**|**rdy**}
a notification should be sent using the target specified in the **XRDPREP_NOTIFY** environmental variable (see the notes). This is only relevant for **prep** and **stage** requests. When **fin** is specified, the notification should be sent whether or not the request succeeded. When **rdy** is specified a notification should only be sent upon success.

-p *prty*
is the request priority with 0 being the lowest possible priority. This is only relevant for **prep** and **stage** requests.

-w Files will be accessed for writing. This is only relevant for **prep** and **stage** requests.

-- Indicates the end of the option list. Subsequent arguments are parameters.

handle
is a unique identifier for the request. It is only relevant for **stage** requests as the client must use this handle to **cancel** and **query** requests. An asterisk (*) indicates that no specific handle has been assigned.

req is the name of the request.

path if a file name relevant to the request (e.g. for **stage** the file that should be brought online). The **evict**, **prep** and **stage** requests require at least one or more files to be specified. Other requests may or may not have specified files.

Notes

1) The **XRDPREP_TID** environmental variable holds the unique real-time identification of the client making the request and can be used to tie back to normal **XRootD** messages that display this identification.
2) To maintain backward compatibility, the program should skip any unknown options. If the unknown option is followed by a token that does not start with a dash, it should be skipped as well. This essentially precludes using option parsers that permute the argument list. Permuting the argument list is unnecessary as the plug-in driver guarantees that all options appear before all parameters and the two are separated by a double dash.

3.14.3 Program Responses

1) Success is indicated by the program ending with a status code of zero. Any other status code is considered a failure.
2) Since non-query requests are executed asynchronously, the client is not notified of any program failure as the client is allowed to resume the moment the request is accepted and either queued or dispatched.
3) **Query** requests are executed synchronously and all output written to **stdout** is returned to the client. However, there is a limit of approximately 2,000 bytes. Any output beyond this limit is discarded and the client is told the result has been truncated.

3.14.4 Program Generated Notifications

When the **–n** option is present, the client has requested notifications and the environmental variable **XRDPREP_NOTIFY** contains how the notifications are to be sent. If the envar is not present or is unspecified, the **–n** option should be ignored. When present it will have the form of:

{**tcp**|**udp**}**://***host*:*port***/**

This indicates that a **tcp** or **udp** message should be sent to *host* at *port*. The **fin** option indicates that notifications are to be sent for all events (i.e. successful and unsuccessful staging events). The **rdy** option indicates the client only wants notification for successful events.

Each notification should end with a new line character. A **udp** message may have more than one notification in it. The format of the message should be:

```
req handle {END | status path [msg]}\n

status:  OK | ENOENT | BAD

req:     cancel | evict | prep | stage
```

Where

req is the name of the request.

handle is a unique identifier for the request as supplied in the argument list.

END indicates that processing completed and no more notification will be sent. This is always the last message to be sent.

status is the ending status:

OK	the request completely successful.
ENOENT	the request failed because the file did not exist.
BAD	the request failed, the message describes the error.

path if the target file name.

msg is an optional message describing the failure.

3.15 tpc

```
ofs.tpc  [allow aparms] [autorm] [cksum type] [echo]
         [fcreds [?]auth =envar] [fcpath path] [logok]
         [require {all | client | server} auth]
         [restrict path] [scan {all | stderr | stdout}]
         [streams num[,max]] [ttl tdflt [tmax]]
         [xfr xmax] [pgm prog]
aparms:  [dn name] [group grp] [host hn] [vo vo]
```

```
ofs.tpc redirect host:port [cgi]
```

Function

Enable and set third-party-copy (**TPC**) parameters.

Parameters

allow requests to copy data in behalf of a client are allowed only if the requestor's authentication information matches the specified values. By default, authentication information, if any, is ignored. One or more values must be specified and the incoming requestor's information must match all of the specified values. Possibilities are:

dn	- distinguished name	**host**	- host name
group	- the unix group name	**vo**	- virtual organization name

autorm

automatically removes the destination file should the copy fail (i.e. return a non-zero return code). By default, the destination file is always left in place regardless of success or failure.

cksum *type*
files copied to the destination host must be check summed and verified using a checksum of the specified *type* (e.g., **adler32**, **crc32**, **crc32c**, **md5**). By default, checksums are computed and verified only if the client requests it.

echo Write to the log file all output subject to the **scan** option. By default, scanned output is not written to the log file.

fcreds [**?**]*auth* **=***envar*
forward client's delegated credentials when authenticating with *auth* protocol (e.g. **gsi**). The credentials are written to a file and the path is set in *envar* prior to starting the copy program. The operation fails if the client does not have credentials that can be forwarded. If the *auth* is prefixed by a question mark (**?**), the server's credentials are used if the client did not supply delegated credentials. See the next secion for more information.

fcpath *path*
specifies the directory path where forwarded credentials are to be written. The default path is "*adminpath*/**.ofs/.tpccreds**" which is qualified with the server's instance name, if any. The directory path is automatically created should it not exist.

logok Write to the log file all successful **TPC** authorizations. By default, only authorization failures are logged.

require {**all** | **client** | **server**} *auth*
allows a **TPC** request only when the requesting client or server have authenticated using the *auth* authentication protocol (e.g. gsi, krb5, etc). Different authentication protocols may apply to a client and server. Specifying **all** requires the same authentication protocol to be used by both. By default, no authentication requirements are applied.

restrict *path*
restricts **TPC** requests to files residing in a path whose prefix matches the specified *path*. By default, no path restrictions are applied.

scan {**all** | **stderr** | **stdout**}

scans the output of the copy command for a possible error message describing a failure. If the copy returns a non-zero return code (i.e. failure) , the scanned error message is returned to the client as the reason. Specifying **stderr** scans only lines written to **stderr**. Specifying **stdout** scans only lines written to **stdout**. The default, **all**, scans **stderr** and **stdout**. See the usage notes describing error message detection. Also see the **echo** option.

streams *num*[,*max*]

the number of TCP streams to use to effect the copy. This can be followed by a comma and *max* which is the maximum number of streams allowed. If the client asks for more streams than *max*, it is set to *max*. The **streams** value is passed to the program using the **'-S'** option only if it is greater than 1. The default number of streams is 1. The default *max* is set to *num* and can be no more than 15.

ttl *tdflt* [*tmax*]

is the default amount of time a client **TPC** authorization may live. You may also specify the maximum amount of time a client may request. Authorizations are automatically canceled when their time to live (i.e. **ttl**) has been exceeded. Each value may be suffixed by **s**, **h**, or **m** (the case is immaterial) to indicate that seconds (the default), minutes, or hours are being specified, respectively. The default *tdflt* is 7 seconds and the default *tmax* is 15 seconds.

xfr *xmax*

the maximum number of simultaneous TPC copies that max occur at the same time. The default is 9.

pgm *prog*

is the program to be executed to copy the file from the source server to the destination server. The **pgm** option must be the last option on the directive line since all remaining characters are used as parameters to the program. The default is **xrdcp**.

redirect

TPC requests issued to a destination server should be redirected to the specified *host*:*port*. See the next section for proper use of this directive.

Defaults

By default, TPC is not enabled. Should **ofs.tpc** be specified, the following defaults are used for unspecified parameters.

```
ofs.tpc ttl 7 15 xfr 9 pgm xrdcp --server [-C type] [-S num]
```

Notes

1) The **ofs.tpc** directives are cumulative. This means you can break up the directive into multiple lines. This allows specifying multiple **allow**, **fcreds**, **require**, and **restrict** parameters.
2) When using delegated credentials, the credentials are temporarily written to disk. For enhanced security you should specify the **fcpath** option to point to a disk partition that uses hardware encryption.
3) When you specify a program (i.e. **pgm** parameter) the program is invoked with any specified arguments possibly followed by a '**-C**' option if the **cksum** parameter is specified or when the client requests checksum processing and two parameters, as follows:

   -C *type*[**:***value*]] *srcurl destpfn*

   Where *type* is the checksum type (e.g. **adler32**, **crc32**, **crc32c**, **md5**) and the optional *value* is the expected checksum value expressed in hexadecimal. The program should be prepared to handle this option. The *srcurl* is the source URL that includes CGI information required by the source server to accomplish the third party copy. The *destpfn* is the physical file name at the local server. This file should be truncated to a length of zero and then over-written. It should not be removed prior to copying as it may be a symbolic link to another physical location.
4) When you specify a program (i.e. **pgm** parameter) the program is invoked with any specified arguments possibly followed by a '**-S**' option if the **streams** parameter is specified with a default value greater than 1 or if the client requests more than 1 stream, as follows:

   -S *numstreams*
5) The **TPC** process always scans the output of the copy command for any relevant error message that can be used to inform the client of the nature of a failure. The last list that contains a colon is considered the most relevant error message. Text after the colon is treated as the reason for the failure. This message is only used when the copy returns a non-zero return code.
6) Refer to the "XRootD Third Party Protocol Reference" for a description of the third party copy (TPC) protocol.

3.15.1 Redirecting TPC requests

You can configure a destination (i.e. the endpoint receiving a file) to redirect **TPC** requests to another host for processing. This means that an **ofs.tpc redirect** directive may apply to a redirector as well as a server. **TPC** redirection may be used in cases where you would like to better manage load as well as other special processing. However, special care must be taken depending on how you handle **TPC** requests.

When you configure **TPC** to use delegated credentials, you may redirect to any endpoint since the request is not tied to any specific server. This is particularly useful because you need not configure every server to generate delegated credentials for each login, only the redirection target server needs to generate delegated credentials and only if it will be actually receiving the file. This will reduce the authentication latency as delegation processing will occur only for **TPC** requests. Information on delegated **TPC** can be found in the next section.

Conversely, when delegation is not used, the **TPC** request may only be redirected to another **xrootd** daemon running on the same host. This is because the request is necessarily tied to the host that will be receiving the file. In this case, you may specify **localhost** as the redirection target hostname.

The ofs.redirect directive also allows you to specify additional **cgi** information that the client will add to the file **URL** used to open the file at redirection target. The redirection target should not have an **ofs.redirect** directive as this may cause a redirection loop.

3.15.2 The TPC Script

The TPC script is free to do anything it wants. However, it should be careful on how it deals with the destination path. This path, many times, is a symlink to the actual data file. Hence, it should never be removed or renamed. To delete space allocated to the file, if any, it should simply truncate the destination file to zero byes. The client tracks copy progress by periodically doing a stat on the destination file. If tha file has been renamed the client will see no progress and eventually terminate the copy operation. If the file symlink is removed and then recreated; it will likely be placed in the wrong partition creating other problems.

For tracing purposes, the envar **XRD_TRACEID** is set to the trace identifier associate with the client that requested the copy. Messages should print this value so it can be paired with the original request.

3.15.3 Third Party Copy Using Delegated Credentials

Third party copy can use delegated credentials to perform the copy for those authentication protocols that support delegation. Currently, only **gsi** authentication supports delegation. Normally, when you specify that delegated credentials are to be used via the **fcreds** option, the client is required to supply such credentials. If the client does not supply delegated credentials, the copy request is rejected. You may specify a fallback by prefixing the auth specified with a question mark. The fallback uses the server's credentials and is backward compatible with earlier versions of **tpc**. Be aware that the server must have usable credentials for the copy to succeed in such a case (e.g. **gsi** proxy credentials).

In order to successfully use delegation three conditions must be met:

1) The client must enable credential delegation.
2) The server's authentication protocol must be configured for delegation.
3) The **ofs.tpc** directive must enable server-side delegation.

Typically, the client enables delegation by using the **delegate** argument of the **–tpc** command line option; for example,

xrdcp -tpc delegate only *sourcefile destinationfile*

Configuring the server's authentication protocol to use delegated credentials is largely dependent on the protocol being used. Since delegation is currently only supported by **gsi** authentication you should specify

sec.protocol gsi -dlgpxy:1 -exppxy:=creds *any_additional_options*

in the configuration file. Refer to the **gsi** protocol in the **Security Reference** for complete information.

Finally, to enable server-side delegation for **gsi** you should specify

ofs.tpc fcreds gsi =X509_USER_PROXY *any_additional_options*

Since delegated credentials are only relevant for the destination server (i.e.. there is no requirement placed on the source server) delegated third party copy securely works with any source server that supports **gsi** authentication even if it does not support the **tpc** protocol. Using a fallback, even though it works, is not secure since authorization to access the source file is based on the server's permissions not the client's permissions.

3.16 trace

```
ofs.trace [-]toption [ [-]toption ] [• • •]
toption: aio | all | chmod | close | closedir |
         debug | delay | dir | exists | fsctl |
         getstats | io | mkdir | most | open |
         opendir | qscan | read | readdir | redirect |
         remove | rename | sync | truncate | write
```

Function

Enable tracing.

Parameters

toption

Specifies 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:

aio	traces asynchronous I/O events
all	selects all possible trace levels
chmod	traces change mode requests
close	traces close file requests
closedir	traces close directory requests
debug	traces internal functions
delay	traces client delays requested by other components
dir	equivalent to **closedir opendir readdir**
exists	traces file stat requests
fsctl	traces file system control requests
getstats	traces statistics inquiry requests
io	equivalent to **read write**
mkdir	traces create directory requests
most	equivalent to **all –aio –read –readdir –write**
open	traces file open requests
opendir	trace directory open requests
read	traces file read requests

readdir	traces directory read requests
redirect	traces client redirections
remove	traces file removal requests
rename	traces file rename requests
sync	trace file synchronization requests
truncate	traces file truncate requests
write	trace file write requests

Defaults

ofs.trace -all.

Notes

1) Tracing occurs at the logical file system level. Additional tracing is available at the physical file system level using the **oss.trace** directive. Other components have additional tracing mechanisms.
2) Enabling tracing can seriously *degrade* overall performance. This directive should *only* be used for debugging purposes..

Example

```
ofs.trace most -dir
```

3.17 xattr

```
ofs.xattr [maxnsz nsz] [maxvsz vsz] [uset {on | off}]
```

Function

Specify file extended attribute parameters.

Parameters

maxnsz *nsz*

specifies the maximum length for extended attribute names. Specify a value between 9 and 255, inclusive. The default is 255.

maxvsz *vsz*

specifies the maximum length for extended attribute value. The default is 65536 (i.e. 64K) and can only be set lower.

uset {**on** | **off**}

specifying on enables user settable extended attributes. This is the default. Specifying off disables user settable extended attributes.

Defaults

```
ofs.usxattr maxnsz 248 maxvsz 65536 uset on
```

Notes

1) The user settable limits apply to all exported file system. You should specify the lowest limit for attribute names and values supported by any exported file system.
2) The **ofs** reserves 8 characters of the extended attribute name for internal purposes. Hence, the user settable limit is 8 characters less than what you actually specify as the **maxnsz** limit.
3) Disabling user settable attributes does not disable internal use of extended attributes.
4) See the **oss.defaults** and **all.export** directives on how to disable extended attribute on a path by path basis for users and internal use.
5) Unspecified values revert to the default setting.

Example

```
ofs.usxattr maxvsz 4096
```

3.18 xattrlib

```
ofs.xattrlib {osslib | [++] path} [parms]
```

Function

Specify the location of the extended attribute interface layer.

Parameters

osslib Specifies that the extended attribute plug-in should be loaded from the library identified by the **ofs.osslib** directive.

++ The specified plug-in should stack on top of the existing plug-in or default. A stacked plug-in cannot be overridden by a subsequent directive.

path The absolute path to the shared library that contains an implementation of the extended attribute interface that **ofs** is to use to fetch and store extended attributes.

parms Optional parameters to be passed to the extended attribute object creation function.

Defaults

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

Notes

1) The extended attribute interface is defined in the **XrdSysXAttr.hh** include file. Refer to this file on how to create a custom extended attribute implementation.
2) See the **ofs.osslib** directive for an alternative way of specifying the location of the extended attribute plug-in.

Example

```
ofs.xattrlib /opt/xrootd/lib/libmyXAttr.so
```

4 Common oss Configuration Directives

4.1 alloc

```
oss.alloc minspace[k | m | g] [ headroom [ fuzz ] ]
```

Function

Specify the way a disk partition is selected for file placement.

Parameters

minspace

The minimum number of free space, in bytes, that must be available in the partition in order for it to be selected. This value also establishes the minimum allocation size for new files. The value may be suffixed by **k**, **m**, or **g** to scale *num* by 2^{10}, 2^{20}, or 2^{30}, respectively. Specifying an asterisk uses the default value.

headroom

The percentage of the requested amount of space to be added to the request in order to compute the allocation size. This can be considered an allocation overhead. Specifying an asterisk uses the default value.

fuzz The percentage difference between two free space quantities that must exist in order for them to be .differentiable. Therefore, quantities that differ by less than the specified percentage are considered equivalent. See the notes for a more detailed explanation of this parameter. Specify a value between 0 and 100, inclusive. Specifying an asterisk uses the default value.

Defaults

```
oss.alloc 0 0 0
```

Notes

1) The *minspace* parameter does not necessarily allow you to leave a certain amount of space free in a partition because when the requested amount equals the *minspace* value and a partition has *minspace* bytes left, the partition may be selected.

2) The *headroom* parameter allows you to effectively overestimate a requested amount of space. If a creation request does not specify the allocation amount, the *minspace* quantity is used.
3) The *fuzz* parameter controls the order in which partitions are selected. At the extremes; zero effectively forces **oss** to always use the partition with the largest amount of free space; while 100 forces **oss** to use a round-robin allocation scheme. Intermediate values proportionately merge the two types of strategies.
4) The case of the scaling letter is not important.
5) The allocation size may be passed to **oss** via opaque information using the **`&oss.asize`** variable. Refer to the section "**Opaque Information**" on how to specify opaque information.

Example

```
oss.alloc 100M 20 50
```

4.2 defaults

```
oss.defaults options

options: cache [no]check [no]dread
         {forcero | readonly | r/o | r/w | [not]writable}
         {inplace | outplace} {local | global | globalro}
         {[no]mig | [not]migratable} [no]mkeep [no]mlock
         [no]mmap [no]purge [no]rcreate [no]stage stage+
         [no]xattrs
```

Function

Specify default file processing options.

Parameters

Option	Disabled/Enabled Function	Default
cache	Designate path as cacheable space. See notes on the implications.	Only when used with **Xcache**.
[**no**]**check**	[Do not] check that a file exists in a remote storage system prior to creating the file in the local file system.	**check** when **migrate** specified with a **r/w** path; otherwise **nocheck**.
[**no**]**dread**	[Do not] read the remote storage system directory contents to list a directory.	**dread** when **rsscmd** specified; otherwise **nodread**.
forcero	Convert all file open requests to read-only access.	**writable**
inplace	Do not use file systems specified via the **oss.space** directive; instead, place file data in the name space.	**outplace**
local	Do not export this path via the cluster manager (i.e., redirector).	**global**
global	Export path via the cluster manager (i.e., redirector).	**global**

Option	Disabled/Enabled Function	Default
globalro	Export path via the cluster manager (i.e., redirector) as an r/o path.	**global**
[no]mig	Local files are [not] migrated to a remote storage system (**[not]migratable** is a synonym).	**nomig**
[no]mkeep	[Do not] keep memory mapped files in virtual memory.	**nomkeep**
[no]mlock	[Do not] lock memory mapped files in real memory.	**nomlock**
[no]mmap	[Do not] memory map files.	**nommap**
outplace	Use file systems specified via the **oss.space** directive for file data.	**outplace**
[no]purge	[Do not] purge little used files.	**nopurge**
[no]rcreate	[Do not] create files in a remote storage system when they are created on local migratable disk	**norcreate**
readonly **r/o**	Files may only be opened for read access.	**writable**
r/w	Path is writable.	**writable**
[no]stage	[Do not] stage a file from a remote storage system should it not exist in the local file system at open time.	**nostage**
stage+	Same as **stage** above but also exports the attribute to meta-managers.	
[not]writable	Path is [not] writable.	**writable**
[no]xattrs	Path does [not] allow extended attributes	**xattrs**

Notes

1) Directive options are applied to paths specified by the **export** directive. This allows you to selectively over-ride the default,
2) The **defaults** directive *should* be specified prior to any **export** directives.
3) Components that use the **all.export** directive also use the **oss.defaults** directive to establish path defaults.
4) Migration and staging is, by default, provided by the **F**ile **R**esidency **M**anager (**FRM**). To use the **FRM** you must configure and run **frm_xfrd**. See the "File Residency Manager Reference" for details.
5) You provide real-time remote storage services using the **rsscmd** directive and an alternate staging implementation using the **stagecmd** directive.

Notes on cache

1) The cache attribute indicates that the path is used for data caching. As such, the path automatically becomes **readonly**.
2) Conflicting attributes are turned off irrespective of what is specified. Thus, the attributes: **notwritable**, **nomig**, **nomkeep**, **nomlock**, **nommap**, **nopurge**, **norcreate**, and **nostage** are enforced.
3) When the Proxy File Cache (**pfc**) component (a.k.a. **Xcache**) configures the Open Storage System (**oss**) for its use, it automatically sets the **cache** attribute for all exported path.
4) When Clustering Management Services (**cms**) see a path with the cache attribute, special processing is done in order to support cached space. It is crucial that the cache option be specified for any export directives target to the **cmsd** during configuration.

Notes on [no]check

1) For data consistency, an identically named file should not exist in a remote storage system when creating a new file or renaming an existing file in the local file system.
2) In order to prevent the creation of an inconsistent set of files; the **check** becomes the default for paths with **migrate** and **writable** attributes. In this case, you must provide an **rsscmd** so that remote file existence can be determined.
3) If you accept the default check attribute or explicitly specify **check** you must also specify the **rsscmd** directive.
4) You may turn off consistency checking by explicitly specifying **nocheck**. A warning may be issued if the **oss** determines that an inconsistency may arise.

Notes on [no]dread

1) The **nodread** option significantly improves performance for most directory lookup operations when the directory resides in a remote storage system (e.g., Mass Storage System).
2) The **nodread** option should only be used on paths where the client is not sensitive to the fact that a directory may appear to have a subset of the actual files that may populate the directory.
3) If you specify **dread** you must also specify the **rsscmd** directive.

Notes on forcero

1) The **forcero** option forces all read/write file open requests to be converted to read/only opens, thus preventing modifications to all files. No errors are returned unless an actual write operation is attempted.
2) When the **forcero** option is specified, files may not be created, deleted, or renamed. Any attempt to do so causes a "read only filesystem" error to be reflected to the client program.

Notes on [no]mig or [not]migratable

1) The **migratable** option informs **oss** that file may be migrated to a remote storage system by some external process. To assist that process, special files are created to coordinate migration of modified files in the local file system to the remote storage system.
2) The word **mig** is a valid abbreviation for **migratabl**e.
3) The word **nomig** is a valid abbreviation for **notmigratable**.

Notes on [no]mkeep

1) The **mkeep** directive implies **mmap** and **forcedro**. Hence, memory mapped files are considered read-only.
2) When **mkeep** is specified, the server attempts to memory map every file that is opened and keep the mapping available even after the file is closed.
3) The default, **nomkeep**, when used with **mmap**, makes the virtual memory space of a memory mapped file available for reuse when the memory mapped file is closed.
4) See **mmap** for additional notes.

Notes on [no]mlock

1) The **mlock** directive implies **mmap** and **forcedro**. Hence, memory mapped files are considered read-only.
2) When **mlock** is specified, the server attempts to memory map every file that is opened and lock the memory space in real memory.
3) The default, **nomlock**, when used with **mmap**, makes the real memory space of a memory mapped file available for displacement when the operating system needs space for more critical functions.
4) The **mlock** option is automatically disabled if the server does not have privileges to lock virtual memory pages in real memory.
5) See **mmap** for additional notes.

Notes on [no]mmap

1) The **mmap** directive implies **forcedro**. Hence, memory mapped files are considered read-only.
2) When **mmap** is specified the server attempts to memory map every file that is opened.
3) The default, **nommap** uses standard file system I/O to bring data into memory.
4) Memory mapped I/O can reduce server overhead by 50% or more. However, it does significantly impact memory usage. Typically, you should only memory map high use random access files.
5) Use the **memfile** directive to further tune memory mapped I/O such as the amount of memory to devote to memory mapped file, and to memory map files on an individual file basis.

Notes on [no]purge

1) This option is used by **frm_purged**, part of the File Residency Manager, to determine which least used files are allowed to be removed from local disk.

Notes on [no]rcreate

1) You ***must*** specify the **rsscmd** directive to provide an access path to the remote storage system when you specify **rcreate**.

Notes on readonly

1) The **readonly** option forces an immediate error to occur should a file be opened for read/write access. Files may also not be created, deleted, or renamed. Any attempt to do so causes a “read only filesystem” error to be reflected to the client program.

Notes on [no]stage

1) When **stage** is in effect and a file is opened but not found on local disk, the **oss** will either use its own built-in staging mechanism or the program specified by the **stagecmd** directive to get a copy of the file. The **nostage** option prohibits this action.
2) The built-in mechanism used by oss to bring files in relies on you running **frm_xfrd**. Refer to the File Residency Manager (**FRM**) reference for more information.
2) The **nostage** is useful in those instances where installation policy mandates that files be manually pre-staged prior to use.
3) The **nostage** and **stage** directives may be applied to selected paths using the **export** directive.
4) Normally, the stage attribute is not exported to meta-managers. If you need the attribute fully exported then specify '**stage+**'.

Notes on [no]xattrs

1) By default, extended attributes are enabled for all files and is used to record various information (e.g. checksums). If an exported path resides on a file system that does not support extended attributes, you should specify **noxattrs** for the path to avoid various error messages appearing in the log.
2) Disabling extended attributes also disables any features that rely on being able to record extended information for a file using extended attributes.
3) Also see the **ofs.xattr** directive about controlling user settable extended attributes.

Example

```
oss.defaults stage forcero
```

4.3 export

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

options: cache [no]check [no]dread

         {forcero | readonly | r/o | r/w | [not]writable}

         {inplace | outplace} {local | global | globalro}

         {[no]mig | [not]migratable} [no]mkeep [no]mlock

         [no]mmap [no]purge [no]rcreate [no]stage stage+

         [no]xattrs
```

Function

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

Parameters

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

xopts Are **xrootd** related export options. These must appear before **oss** path options. Refer to the **xrootd** reference for available **xrootd** options.

options

Are the options to apply to any path whose prefix matches *path*. The options are identical to those allows with the **oss.defaults** directive. See that directive for an explanation of each option.

Defaults

All paths are processed according to the default options in effect at the time the path directive is encountered. Defaults can be set for all of the options (see the **defaults** directive).

Notes

1) Any number of **export** directives may be specified. They are cumulative and are checked in decreasing length order (i.e., most-specific to least specific).
2) Refer to the corresponding **defaults** directive for complete details on the option's effect.
3) The **export** directive is used by the **xrootd** protocol to determine which paths are valid for incoming client requests.
4) The **export** directive is used by clustering services to determine which paths are available via the cluster manager (i.e., redirector).

Example

```
oss.export /xrd/files/staged mig nodread rcreate
```

4.4 localroot

```
oss.localroot path
```

Function

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

Parameters

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

Defaults

None. Paths are used locally as specified.

Notes

1) The **localroot** directive allows you to keep the external namespace consistent even when you move the associated file system from one location to another. This is because the **oss** always internally prefixes the **localroot** to the export path when dealing with file paths.
2) As an example, the exported file system is mounted at **/xrd**. This means that all file paths star with "**/xrd**". The **localroot** directive allows you to remount the file system elsewhere without changing the exported name. So, if now you need to mount the file system at **/usr/xrd** then by specifying
   oss.localroot /usr
   the external view of the file system would remain the same since **oss** automatically prefixes all paths with **/usr** and thus uses the new mount point.
3) Use the **pss.localroot** directive to specify a local root for a proxy server.
4) For default Name2Name plug-ins, the specified local root *path* must exist and be a directory.

Example

```
oss.localroot /usr
```

4.5 remoteroot

oss.remoteroot *path*

Function

Specify where the local file system name space is actually rooted in the remote storage system (e.g., Mass Storage System).

Parameters

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

Defaults

None. Paths are sent to the remote storage system service as specified.

Notes

1) The **remoteroot** directive allows you to place the online file namespace in a different location within the remote storage system. Say that the online file system is exported as **/xrd**. This means that all logical file names start with **/xrd**. If you specify

   oss.remoteroot /usr

   then the file namespace would be rooted at **/usr/xrd** within the remote storage system because all paths would be prefixed by **/usr** before being sent to the remote storage system service.
2) The remoteroot file path may also specify a url. This may allow you to better integrate with various copy programs. For instance, specifying

   oss.remoteroot http://rserver:2080/

   then all paths would be prefixed by the url before being sent to the remote storage system service.

Example

```
oss.remoteroot /usr
```

4.6 rsscmd

oss.rsscmd *command*

Function

Specify the command that communicates with a remote storage service (e.g., Mass Storage System) to perform meta-data operations.

Parameters

command

Command to execute to perform remote meta-data operations (e.g., list, remove, rename, and stat). Specify an absolute path along with the command name.

Defaults

None, you must specify this directive to allow real-time remote storage system access.

Notes

1) When this directive is specified, **check** and **dread** become the default attributes for all directories. See related directives: **rcreate**, **migrate**, and **stage**.
2) This directive is mandatory if **check**, **dread,** or **rcreate** is specified.
3) When the *command* is invoked, the desired operation is passed via the command line, as follows:

create *rmtfn mode*	- create a file using the specified octal mode.
dlist *rmtdir*	- list the contents of a directory.
exists *rmtfn*	- indicate whether or not the file exists[1].
locate *rmtfn*	- if the file exists returns its location.
mv *oldrmtfn newrmtfn*	- rename a file.
rm *rmtfn*	- remove a file.
statx *rmtpath*	- return information about a directory or file.

4) The *rmtdir* and *rmtfn* (above) denote the name of the directory or file in the remote storage system, respectively. This is the logical file name transformed by the **remoteroot** directive or the **namelib** plug-in.

[1] The **exists** and **locate** operation are only passed if the **rsscmd** directive is used. The backward compatibility directive, **msscmd**, does not receive these operations.

5) The *command* must return a numeric return code followed by a newline character; followed by any required output (e.g., a newline separated list of directory entries). A return code of zero indicates that the command succeeded. Non-zero return code values have the same meaning as defined in "errno.h" or its equivalent (e.g., 2 means file not found).
6) The **dlist** operation must respond with a list of directory entries; each entry separated by a newline character. A null line (i.e., one with only '\n') indicates the end of the list.
7) The **statx** operation is issued in response to client stat() requests. The *command* must respond with blank separated tokens of the form:

   ftype mtype nlink uid gid atime ctime mtime size blocksz blocks

   where:

   ftype - is the entry type as the letter d for directory or f for file.
   mtype - is the mode as would have been displayed by the ls command (e.g., rwxr-x---).
   nlink - is the number of hard links to the entry
   uid - is the numeric user id that owns the file.
   gid - is the numeric group id assigned to the file.
   atime - is the Unix time when the file was accessed.
   ctime - is the Unix time when the file was created.
   mtime - is the Unix time when the file was modified.
   size - is the size of the file in bytes.
   blocksz - is the underlying block size of the system.
   blocks - is the number of *blocksz* blocks occupied by the file.
8) The **exists** operation is issued when the **oss** needs to determine the existence of a file. The *command* must respond with a return code 0 if the file exists, 2 if it does not exists, or any other "errno.h" value which indicates that existence cannot be determined.
9) The **locate** operation is issued when the **oss** needs to determine the location of a file. The *command* must respond with a return code 0 if the file exists, 2 if it does not exists, or any other "errno.h" value which indicates that existence cannot be determined. If the file exists, the location of the file should be returned as one or more newline separated host names.

Example

```
oss.rsscmd /usr/local/bin/rsstalk
```

4.7 **space** (*definition*)

```
oss.space name { path | ppfx* } [ chkmount mid [ nofail ]]
```

Function

Specify the location of a file system that is to be used to hold data files.

Parameters

chkmount

Enables disk mount verification. Specify:

mid The arbitrary but distinguished mount identifier of the file system partition that should be mounted at *path*. See the subsequent section on how to use mount verification. Specify a 1- to 63-character name.

nofail specifies that server initialization should not fail if the mount cannot be verified; otherwise, the server terminates at the end of initialization. In either case, the space associated with *path* is not added to the list of allowable places where new files can be placed.

name The arbitrary logical name for the file system. Specify a 1- to 63-character name.

path The absolute path in the file system to be used.

*ppfx** All directory entries that start with *ppfx* in the containing directory are to be used to hold data files.

Defaults

None. Multiple file systems are not supported unless the **space** directive is specified.

Notes

1) The **oss space** directive allows you to concatenate multiple file systems (i.e., physical disk partitions) into a single file system. In some sense, this is an implementation of a "poor man's" volume manager.

2) If the underlying file system includes a volume manager, there may be no reason to use this feature. The volume manager should already support disk partition aggregation. On the other hand, the **oss** implementation provides fine grained control of disk partitions because it externalizes the support. For instance, clients can logically direct allocation requests to specific partitions for performance reasons; something not available in most volume managers.
3) File systems need not be physical partitions. When different directories on the same physical partition are specified, they are treated as different logical partitions from a space management viewpoint. This allows you to create arbitrary views of available space (e.g., by **SRM** static space token).
4) Each *path* must be associated with a logical *name*. Any number of paths may be associated with the name and a path may be assigned to any number of names by listing the path each time with a different *name*.
5) It is highly recommended that you use star (*) notation for *path*. This allows you to mount additional file systems without the need of updating the configuration file.
6) The *name* is used to direct space allocation. A request may specify the name of the space where a file is to be allocated. The **oss** will only allocate space using one of the paths associated with the specified name.
7) In the absence of any specified space name, **oss** uses the name "public". Unless, you associate at least one path with the name "public", file creation will fail in the absence of a name specification.
8) The space name is passed to **oss** via opaque information using the **`&oss.cgroup`** variable. Refer to the "Opaque Information" chapter in this reference for information on how to specify opaque information.
9) Once space is defined using **space** directives, files are always allocated within the set of specified paths. You may create exceptions and allocate files in the file system holding the name-space by specifying the **inplace** option on the **export** directive for specific exported paths.
10) Clustering services use the **oss.space** directive to identify writable file systems. These file systems are monitored for space availability on behalf of the cluster manager (i.e., redirector).
11) See the oss.space directive for space assignment for additional options.

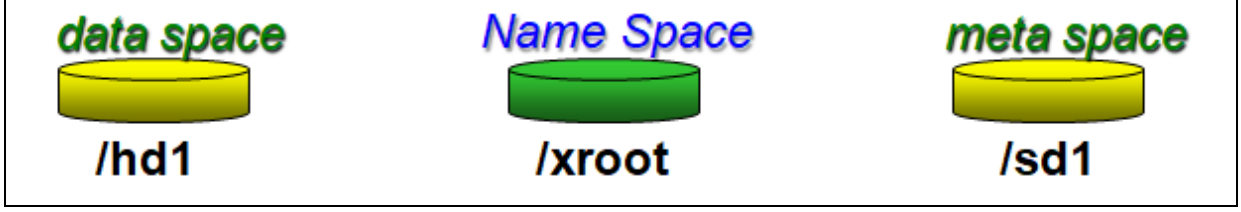
Example

```
oss.space public /xrootd/fsdev01
```

4.7.1 Oss Space Explained

Consider the following configuration snippet:

```
oss.localroot /xroot
oss.space data /hd1
oss.space meta /sd1
```



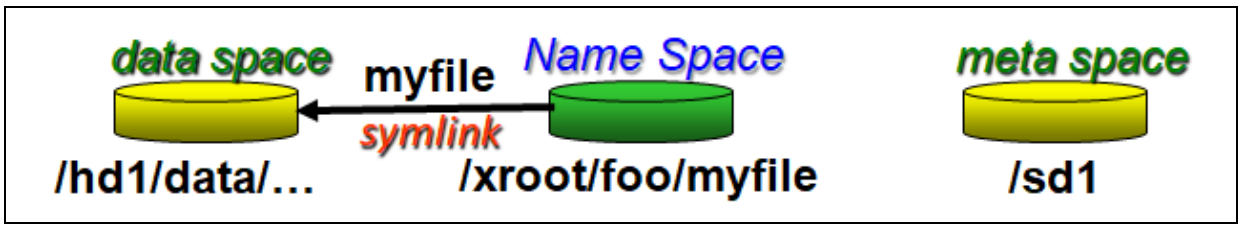
Three mounted file systems have been described:

- The file system to be used for file paths (i.e. the name space) mounted at **/xroot**,
- The file system to be used for data files mounted at **/hd1** (a hard drive), and
- The file system to be used for metadata files mounted at **/sd1** (an ssd drive).

Assume a client creates a new file using the path specification

```
/foo/myfile?oss.cgroup=data
```

Then the following symbolic link and file would be created:

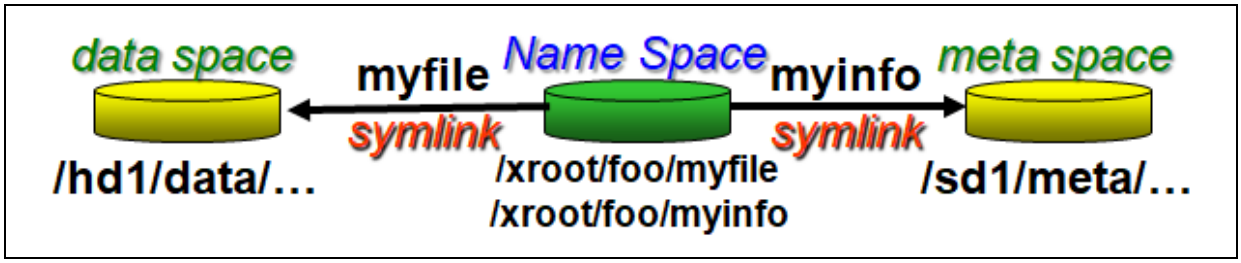


The name space partition contains a symbolic link that points to the location of the actual data file in the **/hd1** partition. The path and name used in the data space is specially formulated to prevent overloading any single directory. The path in the name space is the client specified path prefixed by *localroot*. The client always refers to the file using the original path. The **oss** is responsible for arranging the location of the path and actual data to correspond to the space the client wanted to use.

Similarly, should the client create a new file as

```
/foo/myinfo?oss.cgroup=meta
```

Then the following symbolic link and file would be created



In this example, spaces have been used to establish high performance space (i.e. ssd) and low performance space (i.e. hard drive) and provide the ability for the client to choose which space to use without sacrificing a single uniform name space. Spaces can be used for any number of reasons like piecing together multiple file systems to provide a large pool of storage space that can grow as needed.

You can use the **oss.space** directive to specify defaults and specific spaces that must be used based on path. This is described in the subsequent section.

4.7.2 Using Mount Verification

If the **oss.space** *path* is a mount point for a volume it is usually a good idea to verify that the expected volume is actually mounted at that directory (i.e. on the last component of *path*). While not frequent, volume mounts might not occur for a variety of reasons and in a containerized or virtual machine environment start-up mistakes may virtually mount the wrong volume. Should this happen, new files will be placed in the wrong location or it may be possible, though highly unlikely, that the wrong files may be opened for reading.

The **chkmount** option provides a mechanism to verify that the expected volume has, in fact, been mounted at the designated directory. This is done by verifying that the combination of the mount identifier, *mid*, suffixed with a dot and the mount point directory name exist as a filename entry in the *path* directory assigned to the space.

You should always choose a distinguished name for *mid*. It should not be reused for other servers. If running in a containerized environment it should not be the name of the underlying physical host as that may change and lead to confusion. Generally the *mid* should correspond to the specific service being offered using the group of mounted volumes. As an example, let's use a mount identifier **atlas_Xcache01** in the following **oss.space** directive:

```
oss.space scratch /scratch/vol1 chkmount atlas_Xcache01
```

When the directive is encountered the file **/scratch/vol1/atlas_xchace01.vol1** must exists in order for the space to be accepted. If it cannot be found, no matter the reason, mount verification fails. You may, of course, specify the **nofail** option to prevent a wrong disk partition from being used but otherwise allow read-only access to existing data. The log always contains messages about such kind of problems.

This means there are a few simple steps that need to be done to make this work:

1) Come up with a distinguished name,
2) Bring up the server,
3) Verify that the correct volumes are actually mounted (the "**df –h**" command displays mounted volumes).
4) Use the touch command to create a file using the naming scheme explained above in each mount point that you will be using.
5) You may wish to make the file only readable to prevent inadvertent deletion.

4.8 space (*assignment*)

```
oss.space name { assign | default } lfnpfx [lfnpfx [. . .]]
```

Function

Specify the space name based on file creation path.

Parameters

name The logical name for the file system. Specify a 1- to 63-character name for an existing space definition.

assign The space name is assigned to the corresponding *lfnpfx* even if the client specified another space name using the **oss.cgroup CGI** element.

default

The space name is assigned to the corresponding *lfnpfx* should the client not specify a space name using the **oss.cgroup CGI** element.

lfnpfx is a logical file name prefix. When a file is created with this prefix it is assigned or defaulted to the specified space *name*.

Defaults

If the client does not specify a space name when creating a file, the name defaults to **public**.

Notes

1) The order of **oss.space** definition and assignment directives are immaterial. File paths are always checked using the most-to-least specific definition.
2) A warning is issued if you specify a space assignment for a space that has not been defined. If such cases, the client may receive a "file not found" error when creating a file in a non-existent space.

Example

```
oss.space foobar assign /usr/barfoo
```

5 Esoteric oss Configuration Directives

5.1 fdlimit

```
oss.fdlimit fence [ max ]
```

Function

Specify how file descriptors are allocated.

Parameters

fence The highest file descriptor number which is to be associated with sockets. File descriptors above this value will be associated with files. Specify a value from zero to *max*. Specifying an asterisk uses *max*/2 as the fence value.

max The highest allowed file descriptor number to be used. If not specified, the current *soft* limit is used. If you specify the word **max** then the current *hard* limit is used. If you do specify a numeric value it must be greater than *fence* or 64, whichever is greater, and less than or equal to the current hard limit. If it exceeds the current hard limit, it is set to the hard limit.

Defaults

```
oss.fdlimit *
```

Notes

1) It is highly recommended that socket descriptors be partitioned from file descriptors. This partitioning substantially improves performance in most operating systems. The default accomplishes this task.

Example

```
oss.fdlimit * max
```

5.2 maxsize

```
oss.maxsize num[k | m | g]
```

Function

Specify the maximum size of a file.

Parameters

num The maximum number of bytes that a file may have. The quantity may be suffixed by **k**, **m**, or **g** to scale *num* by 2^{10}, 2^{20}, or 2^{30}, respectively.

Defaults

None. A file may be as large as the underlying file system allows.

Notes.

1) The case of the scaling letter is not important.

Example

```
oss.maxsize 2G
```

5.3 memfile

```
oss.memfile parms

parms:   [check] [max msz[k | m | g | %]] [off] [preload]
```

Function

Specify memory mapped I/O options.

Parameters

check Specifies that memory mapped I/O should be controlled on an individual file basis using checking the files' extended attributes.

msz The amount of real memory to devote to memory mapped files. Either specify a fixed number of bytes, optionally suffixed by **k**, **m**, or **g** to scale *msz* by 2^{10}, 2^{20}, or 2^{30}, respectively; or number between 1 and 1000, inclusive, suffixed by **%** to indicate that a percentage of real memory should be devoted to memory mapped files. The default is 50% (i.e., half the real memory of the machine).

off Disables memory mapped I/O regardless of other specified directives or parameters.

preload

Pre-loads the file into memory when the file is first opened. The default is to only load the pages actually requested from the file (i.e., demand load).

Defaults

```
oss.memfile max 50%
```

Notes

1) As memory mapped files are opened, the server attempts to find sufficient virtual memory address space for the complete file without exceeding *msz*. If possible, it reclaims address space by un-mapping files that are no longer in use and not marked with the keep attribute. If there is still insufficient address space available, file I/O proceeds using standard file system calls.

2) You may over commit real memory via the max parameter. Be aware that severely over committing memory may lead to a significant increase is I/O to swap space.
3) Memory mapped files are considered to be read-only. The server automatically adds the **forcedro** attribute to any memory mapped file.
4) The **mkeep**, **mlock**, and **mmap** attributes on the **path** directive over-ride the **check** parameter. That is, the server does not check file attributes for any file that is eligible to be memory mapped, locked, and/or kept via the **path** directive.
5) Memory locking is automatically disabled if the server does not have privileges to lock virtual memory pages in real memory.
6) The **preload** parameter should be used if most of the file is likely to be referenced. Do not specify preload when memory is overcommitted as this causes significant I/O thrashing.
7) Files are preloaded once at open time. If the file is closed and re-opened at a later time, another preload occurs. Displaced pages after preloading are brought into real memory on a demand basis.

Example

```
oss.memfile max 200% check map
```

5.4 namelib

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

Function

Specify the location of the file name mapping interface layer.

Parameters

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

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

Defaults

A built-in minimal internal implementation driven by the **localroot** and **remoteroot** directives is used.

Notes

1) The Name2Name interface is defined in XrdOucName2Name.hh include file. Refer to this file on how to create a custom file name mapping algorithm.
2) The Name2Name interface is also used by clustering services.
3) Use the **pss.namelib** directive to specify a Name2Name plug-in for a proxy server.

Example

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

5.5 spacescan

```
oss.spacescan num[s | m | h]
```

Function

Specify how frequently internal statistics are reconciled with actual available space in each disk partition.

Parameters

num The amount of time between disk partition scans. The value may be suffixed by **s**, **h**, or **m** (the case is immaterial) to indicate that seconds (the default), minutes, or hours are being specified, respectively.

Defaults

```
oss.spacescan 600
```

Notes

1) The **oss** periodically scan all disk partitions to make sure that internal statistics about the partition match those reported by the associated file system. The **spacescan** directive controls how frequently this is done.
2) Lower **spacescan** *seconds* values increase **oss** overhead because the scan occurs more frequently.
3) Usage statistics are also reconciled with external systems at the end of each scan.
4) You can also disable space scanning by setting the environment variable **XRDOSSCSCAN** to a value of **off**.

Example

```
oss.spacescan 800
```

5.7 stagecmd

```
oss.stagecmd [async | sync | creates] [|]command
```

Function

Specify the command that brings files from the remote storage system into the local file system.

Parameters

async Indicates that *command* will notify **xrootd** via a named pipe when the staging request completes, along with how it completed. The client is told to wait until the notification is received.

sync Indicates that the client must poll for the completion staging request (i.e., the file appears or it's '**.fail**' counterpart). The client is told to wait for a fixed amount of time and then retries to open the file.

creates

Routes file creation requests to *command*. The *command* is responsible for creating the file.

command

The command to execute to perform a staging or create operation. Specify an absolute path along with the command name. Preceding *command* with an "or" bar designates a long running command that takes its input via standard in (see the notes).

Defaults

The built-in interface to the File Residency Manager's **frm_xfrd** is used.

Notes

1) If you wish to use a remote storage system, have specified the **stage** directive, *and* your processing requirements cannot be met by the File Residency Manager, you must specify the **stagecmd** directive.
2) The **stagecmd** is disabled unless at least one exported path has the **stage** attribute.

3) When the command is specified without a leading or bar, *command* is executed every time a file must be brought in from the remote storage system. The relevant actions are:
    1) The **stagecmd** is executed in a forked process and should either copy the specified file from the remote storage system to a specified location in the local file system, or it should fail.
    2) Success is assumed when the **stagecmd** returns a zero status code. Otherwise, failure is assumed.
    3) The **stagecmd** is invoked via **execve()** system call using the current environment and is passed two arguments. The first argument is the name of the file as it should be known in the Remote storage system. This is the file that must be copied into the local file system. The second argument is the name of the file, as it should be locally known (i.e., the target location and name). Both names specify absolute paths.
4) When the command is specified with a leading or bar, command is executed once and special requests are written to its standard in every time a file must be brought in from the remote storage system. See the **stagemsg** directive on message format details.
5) Should the staging operation fail, the *command* must create a "fail" file; which is a zero-length file whose name is identical to the requested *path* with a "**.fail**" suffix. Failure to do so causes the failing request to be resubmitted.
6) The *command* is normally started once. However, if *command* exits, it is automatically restarted.
7) When **async** is specified, command must send a notification to a named pipe indicating that the staging operation succeeded or failed. The path name of the pipe is contained in the **XRDOFSEVENTS** environmental variable. Notification is sent as a newline terminated ASCII text message. The format is:

stage {**OK** | **ENOENT** | **BAD**} *path* [*msg*] **\n**

OK the staging request successfully completed.
ENOENT the staging request failed because the file could not be found.
BAD the staging request failed for some reason.
path is the full logical path to the file being staged.
msg an optional error message that describes the nature of the failure.

Example

```
oss.stagecmd /usr/local/bin/StageIn
```

5.8 stagemsg

```
oss.stagemsg msgline

msgline: [text] [var] [msgline]

var:     $CGI | $LFN | $PFN | $RFN | $LFN2 | $PFN2 |
         $RFN2 | $FMODE | $HOST | $OFLAG | $TID |
         $USER | $eVar
```

Function

Specify the message to be sent to a piping **stagecmd** when a staging request is received.

Parameters

text Arbitrary text.

var A variable whose value is determined by the current request setting. The following variables may be specified:

$CGI	all of the opaque information specified after the question mark in the file path
$LFN	the logical file name
$PFN	the physical file name as modified by **localroot** or the **namelib** function
$RFN	the remote file name as modified by **remoteroot** or the **namelib** plug-in
$LFN2	the *second* logical file name in a **rename** operation.
$PFN2	the *second* physical file name a rename operation as modified by **localroot** or the **namelib** function
$RFN2	the *second* remote file name a rename operation as modified by **remoteroot** or the **namelib** plug-in
$FMODE	the octal mode associated with a file **chmod**, **create**, and **mkdir** requests
$HOST	the client's host name

$OFLAG	A character sequence describing which file open processing flags are in effect: **a** – O_APPEND **t** – O_TRUNC **c** – O_CREAT **w** – O_WRONLY | O_RDWR **r** – O_RDONLY **x** – O_EXCL
$TID	the client's trace identity
$USER	the authenticated client's name
$*eVar*	any variable that has been passed along with the file name as opaque information

Defaults

The default message is described in the following section.

Notes

1) Variables must begin with a $ (dollar sign) and end with a non-alpha-numeric character.
2) To include a dollar sign into the message, escape it with a back slash ("\").
3) A backslash escape is only recognized when followed by a dollar sign.
4) Important! The **stagemsg** *msgline* is ***not*** subject to general **set** variable substitution.
5) Except for **$CGI**, the implicit value of a variable that has not been set is the variable name itself, including the dollar sign.
6) For **$CGI**, if no opaque information is found, the variable is substituted with the null string.

Example

```
oss.stagemsg stage $LFN $PFN
```

5.8.1 Default Stage Request Message (**stagemsg**)

The default[2] message that is sent to the **stagecmd**'s stdin when a stage operation is required has the following format:

+ *requestid npath priority mode path* [*path* [. . .]]

Where:

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

npath is the notification path to be used to indicate how the request complete. This field may contain:

-	no notification is to be sent.
mailto://user	send e-mail to *user*
tcp://rhost:port/*msg*	send *msg* via tcp to rhost:port
udp://rhost:port/*msg*	send *msg* via udp to rhost:port

priority is the request's priority as a number 0 through 9, inclusive. Zero is the lowest priority, while 9 the highest.

mode is the processing mode and may contain a combination of the following letters:

- **f** send fail notice (not affected by q flag)
- **n** send success notice
- **q** suppress default failure notice (i.e., quiet)
- **r** file is epected to be only read
- **w** allow the file to be modified

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

[2] This message may be specified by using the **stagemsg** directive.

Notes

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

Successful:	**ready**	*requestid msg path*
Unsuccessful:	**unprep**	*requested msg path*

requestid	is the request identifier associated with the completed request.
msg	is the text that followed the notification url (see above). This text is sent without inspection.
path	is the absolute *logical* name of the file that successfully prepared or whose preparation failed.

5.8.2 The Stage Cancel Message

The following message is sent to the **stagecmd**'s stdin to cancel a stage operation:

- *requestid*

Where:

requestid The request identifier used in a previous stage request. All entries with this *requestid* should be removed.

Notes

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

5.8.3 The Stage Query Message

The following message is sent to the **stagecmd**'s stdin to cancel a stage operation:

?

Notes

1) The *command* should respond with a list of new-line separated paths associated with queued requests.
2) You cannot change the format of a stage query request message.

5.9 statlib

```
oss.statlib [options] path [parms]

options: [-arevents] [-non2n] [-preopen]
```

Function

Specify the location of the file system **stat()** function interface layer.

Parameters

-arevents

Calls the plug-in when names are added or removed from the system. This option only applies to the **cmsd**'s with server roles. Include file **XrdOssStatInfo.hh** details the calling conventions.

-non2n

Bypasses the name-to-name translation when calling the **StatInfo()** function in the shared library.

-preopen

The stat function in the library must be called prior to opening a file in addition to being called whenever file state information is needed.

path The absolute path to the shared library that contains an implementation of the **stat()** function interface that **oss** is to use to obtain file state information.

parms Optional parameters to be passed to the **stat()** object creation function.

Defaults

The kernel implementation of **stat()** is used.

Notes

1) The **stat()** function interface is defined in **XrdOssStatInfo.hh** include file. Refer to this file on how to create a custom **stat()** function.
2) The **stat()** interface is also used by clustering services.
3) The file **XrdOssSIgpfsT.c**c is contains an alternate implementation of the **stat()** function and can serve as an example on how to implement a custom function.
4) The **–arevents** option is intended to be used by configurations that implement a constructed namespace (e.g. **SSI**). Such configurations may need notification at the **cmsd** level when the companion **xrootd** has added or removed an entry from the system's namespace.

Example

```
oss.statlib /opt/xrootd/lib/libXrdOssSIgpfsT.so
```

5.10 trace

```
oss.trace [-]toption [ [-]toption ] [• • •]

toption: all | debug | open | opendir
```

Function

Enable tracing at the **oss** level.

Parameters

toption

Specifies 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 except **debug**
debug	traces internal functions
open	traces file open requests
opendir	trace directory open requests

Defaults

ofs.trace -all.

Notes

1) Trace output is currently routed to standard error.
2) Tracing occurs at the physical file system level. Additional tracing is available at the logical file system level using the **ofs.trace** directive.
3) Enabling tracing can seriously *degrade* overall performance. This directive should *only* be used for debugging purposes.

Example

```
ofs.trace all -dir
```

5.11 usage

```
oss.usage parms

parms:    [nolog | log path [sync snum]] [qoutafile file]
```

Function

Specify the handling of usage information.

Parameters

nolog does not log any usage information.

log *path*

logs usage information in the directory identified by *path*.

sync *snum*

synchronizes the **log** *path* file media with memory based usage counters every *snum* changes. Specify a number between 1 and 32767, inclusive. See the usage notes about the default when **log** is specified.

quotafile *file*

indicates that quota information can be found in *file*.

Defaults

```
oss.usage nolog
```

Notes

1) The **usage** directive allows you to save usage information via the **log** parameter across server restarts. Since the information becomes stable, it can be used to enforce quotas.
2) When the **log** parameter is specified, usage information is always written to actual media whenever it changes to maintain a high degree of reliability (i.e. **sync 1**). This may impact performance under heavy file creation loads. The **sync** parameter allows you to reduce the number of times the usage information is synchronized with physical media. Since usage information is always synchronized with physical media every **spacescan** interval, the values are written to physical media even if adjustments are not frequent; providing you some leeway.

3) Even when usage information is written to physical media it may still drift from the true value because of hardware and software failures. You can always determine if the values are correct by running

   frm_admin -c *config* [**-n** *instance*] **audit usage**

   on the server in question. You can also correct the usage information on a running system using the same command:

   frm_admin -c *config* [**-n** *instance*] **audit -fix usage**

   Adding the **-force** option bypasses the confirmation prompt. When usage is corrected, the server updates its internal information within the **spacescan** interval.
4) The quota file is automatically reprocessed should it change. A check for changes is made every **spacescan** interval.
5) Currently, quotas are only reported via the extended attribute query interface. Quotas are not enforced by **xrootd** or **cmsd**.
6) Use high reliability space for *path* and *file*; preferably solid state media.

Example

```
oss.usage log /var/spool
```

5.12 xfr

```
oss.xfr   [options] [{thrds | *}]

options:  [deny dt[s|m|h]] [fdir path] [keep kt[s|m|h]]
          [up]
```

Function

Specify dynamic staging characteristics.

Parameters

dt The time, after a failing stage request for file *x*, future stage requests for file *x* check for the presence of a ".fail"file to determine whether or not to honor the request. After the *dt* period, the staging request for file *x* is retried whether or not a ".fail" file exists. The value may be suffixed by **s**, **h**, or **m** (the case is immaterial) to indicate that seconds (the default), minutes, or hours are being specified, respectively. A value of zero disables this processing so that the presence of a ".fail" file always suppresses the staging of the file.

path The location of the shadow directory where ".fail" files are to be written. The default is to write them along side (i.e. in the same directory) as the failing file. The *path* must be less than or equal to 256 characters.

kt The amount of time a pending stage request should be remembered. During this time, requests for the same file do not create additional entries in the pre-stage queue. In other words, only one additional entry for the same file may be added to the pre-stage queue each *kt*.

up allows client to specify staging priority via opaque information. By default, client-specified staging priority is ignored.

thrds The number of threads to be devoted to staging operations. The value effectively controls the number of staging operations that can occur at any one time. This value is only meaningful for synchronous staging. Specify an asterisk to use the default value.

Defaults

```
oss.xfr deny 3h keep 20m 1
```

Notes

1) The **oss** first checks for the existence of a file whose name is the same as the one being requested but with a suffix of '.fail'. Should such a file exist, the staging operation is not tried and an error is returned to the client. This effectively prevents staging loops.
2) Clients specify staging priority using the OPID "oss.sprty" with the file name presented at open time. Refer to the "Specifying Opaque Information" section in this reference for information on how to specify opaque information.
3) The **localroot** value does not apply to the **fdir** *path*. You must specify and existing real path.

Example

```
oss.xfr deny 1h fdir /tmp/ossfail/
```

6 Enabling Mult-Tiered Storage (MTS)

The **oss** includes **M**ulti-**T**iered **S**torage (**MTS**) support where files can reside locally or remotely elsewhere (e.g., in a Mass Storage System or even some local but slow device). Depending on the configuration, files can be automatically copied from another location to local disk when they are opened.

By default, **MTS** support is not enabled. You enable this support using the **stage** attribute on the **oss.defaults** or **all.export** directive. Other **export** attributes: **check**, **dread**, **migrate**, and **rcreate**, as well as the **oss.remotepath**, **oss.rsscmd**, and **oss.stagecmd** directives modify **MTS** behavior. The **oss** provides built-in **MTS** support via the **F**ile **R**esidency **M**anager (**FRM**) using **frm_xfragent**. To fully use this mechanism, you need to configure and run **frm_xfrd**. Refer to the "File Residency Manager Reference" for details. Alternatively, use the **oss.stagecmd** directive to provide a custom implementation.

When **MTS** support is enabled, special files are created in the file system to coordinate the staging files from an external location as well as migrating files to an external location.

The **oss** layer allows the use of practically any remote storage system to provide multi-tiered storage. The requirements are minimal and include:

- A Unix-like name space,
- a command to transfer data to and from the remote storage system, and
- a command to obtain meta-data about a file in the remote storage system if you wish to provide real-time responsiveness.

There are only two steps for interfacing remote storage with the **oss**. These are:

- To provide a real-time interface, decide on the command to use to obtain remotely maintained meta-data (e.g., file status and directory contents) as well as perform meta-data oriented functions (e.g., remove, rename, etc.). The **rsscmd** directive specifies the command. The arguments and expected responses are documented in the configuration section for the directive.

- To provide a non-default remote storage service interface, decide on the command to use to get a file from remote storage. The **stagecmd** directive specifies the command. The arguments and expected responses are documented in the configuration section for the directive.

6.1 Special MTS Files

The oss component either creates or recognizes meta-files with special suffixes in the name space. Consequently, these files cannot be used to hold data. The following table lists these special files.

File	Origin	Purpose
fn.anew	Created*!	Used while importing *fn* from an external location.
fn.fail	Created*!	Controls future imports and exports of *fn* when a previous such attempt failed. The file may contain details about the failure.
fn.map	---	Reserved for future use.

Special MTS Name Space Files

7 Opaque Information

This section describes the opaque information directives (**OPID**s) recognized by the **ofs** and **oss** layers. A client program creates the opaque information and appends it to a path name used as the target of an open request. The information is opaque in the sense that only the layer to which the information is targeted interprets the information. All opaque information is structures in the same way, as follows:

```
path&layer.directive=arg[,arg[,···]][&layer.directive=···]
```

Where:

path The path passed as an argument to the **kXR_Open** request.

layer The layer to which the directive is sent. Valid layer names are:

ofs the logical file system layer

oss the physical storage system layer.

directive

The name of the specific directive. Directives are documented in the following sections.

arg Directive-specific arguments.

Notes

1) Unrecognized layer names or directive names are ignored.
2) Currently, the **ofs** layer has no opaque directives.
3) Invalid values or arguments to a recognized directive normally result in termination of the request.
4) An **OPID** consists of a value-directive instance. Any number of **OPID**s may be strung together. The **OPID** order is immaterial. However, the last duplicate **OPID** always takes precedence.

Example

```
&oss.cgroup=index&oss.asize=120000000
```

7.1 ofs OPIDs

7.1.1 lcl

```
&ofs.lcl=t
```

Function

Request that an action only occur on the local copy of a file.

Arguments

t Restricts file system operations to the local copy.

Defaults

Defaults are defined by the operation being requested which may involve more than just the local copy the file.

Notes

1) Currently, the **ofs.lcl** OPID controls file deletion. When the OPID is specified along with the file to be removed, only the local copy of the file is removed. Remote copies (e.g., in a Mass Storage System), if any, are not affected.

Example

```
&ofs.lcl=t
```

7.1.2 posc

```
&ofs.posc=t
```

Function

Request **POSC** processing from the **ofs** layer for a new file.

Arguments

t Enables **POSC** processing for the associated file.

Defaults

Default processing mode is specified by the **ofs.persist** directive.

Notes

1) POSC processing may also be enabled by setting the **kXR_posc** option in the **kXR_open** request.
2) Currently, **POSC** processing is enabled regardless of the value given in the **ofs.posc OPID**. For future compatibility, you should always use the single letter '**t**' as the value to enable **POSC** processing.

Example

```
&ofs.posc=t
```

7.2 oss OPIDs

7.2.1 asize

```
&oss.asize=bytes[k | m | g]
```

Function

Provide the **oss** layer the estimated allocation size for a new file.

Arguments

bytes The estimated number of bytes that a newly created database file will need. The value may be suffixed by **k**, **m**, or **g** to scale *num* by 2^{10}, 2^{20}, or 2^{30}, respectively.

Defaults

Default allocation for new databases is determined by the **oss.alloc** configuration directive.

Notes

1) The **alloc** directive provides a hint to the server so that it can more effectively perform space allocation. Like all hints, the server may choose to ignore it.

Example

```
&oss.assize=500m
```

7.2.2 cgroup

```
&oss.cgroup=spacename
```

Function

Specify the name of the space for a newly created file.

Arguments

spacename

the name of the space where the newly created file is to be placed.

Defaults

&oss.cgroup=public

Notes

1) Named spaces allow files to be segregated from one another for arbitrary reasons. Space are defined using the **oss.space** directive.
2) The **cgroup** OPID provides a hint to the server so that it can more effectively perform space allocation. Like all hints, the server may choose to ignore it.
3) The administrator defines what a particular space name means. For instance, the name of the space may correspond to an SRM static space token.

Example

&oss.cgroup=fast

7.2.3 lcl

```
&oss.lcl=1
```

Function

Request that an action only occur on the local copy of a file.

Arguments

1 Restricts file system operations to the local copy.

Defaults

Defaults are defined by the operation being requested which may involve more than just the local copy the file.

Notes

1) Currently, the **oss.lcl** OPID controls open and stat requests. When the OPID is specified along with the file to be open or quried, only the local copy of the file is considered. Remote copies (e.g., in a Mass Storage System), if any, are not considered.

Example

```
&ofs.lcl=t
```

7.2.4 sprty

```
&oss.sprty=priority
```

Function

Specify the staging priority should the database need to be staged to disk.

Arguments

priority

a value from 0 through 15; with 0 being the lowest priority and 15 being the highest priority.

Defaults

```
&oss.sprty=1
```

Notes

1) The **sprty OPID** is ignored unless it is enabled via the **up** option of the **oss.xfr** directive.
2) The specified priority is effective only relative to a policy driven system priority. That is, should your request be assigned a system priority of 4, then the **sprty** is only relative to other requests assigned the same system priority.
3) The **sprty** directive provides a hint to the server so that it can more effectively perform staging. Like all hints, the server may choose to ignore it.
4) The valid range of staging priorities for the File Residency Manager range from 0 (the lowest) to 2 (the highest).

Example

```
&oss.sprty=3
```

8 Virtual Extended Attributes

The **ofs** and **oss** support extended attributes that may be obtained using the getxattr() call available as part of the POSIX support included with **xrootd**. The getxattr function syntax is:

```
ssize_t getxattr(const char *path, const char *name,
                 void *value, size_t size);
```

Operating system man pages document the getxattr function. The specific name attributes that are currently supported are:

Name	Information Returned
xroot.space	Space information relative to *path* and possible **cgroup** specification.
xroot.xattr	Extended attributes associated with *path*.

For each name, information is returned as a cgi string; as described in the following sections.

8.1 Information returned for xroot.space

```
oss.cgroup=cgrp&oss.space=space&oss.free=free&oss.maxf=maxf
&oss.used=used&oss.quota=quota
```

Values

cgrp the space name associated with *path* in the absence of a **oss.cgroup** specification. When **oss.cgroup** is specified with *path*, the specification is returned.

space total number of bytes accessible to *cgrp*.

free number of *space* bytes that are available for possible allocation.

maxf maximum number of contiguous *free* bytes available.

used number of bytes in use (i.e., allocated) by *cgrp*. See notes for caveats.

quota maximum number of bytes *cgrp* is allowed to have allocated. If the value is negative, no quota applies.

Notes

1) If **oss.cgroup** is specified as opaque information with path, then information is returned for the specified space. Otherwise, information is returned relative to where path would have been allocated.
2) Information is returned without any breaks.
3) Since tokens may be returned in an arbitrary order, they should be parsed relative to their **cgi** name.

8.2 Information returned for xroot.xattr

```
oss.cgroup=cgrp&oss.type=type&oss.used=used&oss.mt=mt

&oss.ct=ct&oss.ct=ct&oss.at=at&oss.u=usr&oss.g=grp

&oss.fs=fs&ofs.ap=privs
```

Values

cgrp the space name associated with *path* If *cgrp* is an asterisk (*), then *path* is not online and may exist in any space once it is retrieved from a Remote storage system.

type type of path as a single letter:
d – Directory **f** – File **o** – Other (not file or directory).

used size of the entry in bytes.

mt the entry's Unix modification time.

ct the entry's Unix state change time.

ct the entry's Unix state change time.

usr the name of the entry's owner. An asterisk indicates that the entry is owned by the server.

grp the name of the entry's Unix group. An asterisk indicates that the entry is assigned to the server's primary group.

fs the containing file system characteristics as a single letter:
r – Read/Only file system **w** – Writable file system

privs the requestor's privileges relative to path as a sequence of single letter privileges:

a – all privileges	**l** – lookup a file (i.e., search directory)
d – delete (i.e., remove)a file	**n** – rename a file
i – insert (i.e., create) a file	**r** – read a file
k – lock a file (not used)	**w** – write a file

Notes

1) The **oss.cgroup**, if specified, is ignored.
2) Information is returned without any breaks.
3) Since tokens may be returned in an arbitrary order, they should be parsed relative to their **cgi** name.

9 Document Change History

23 Feb 2005

- Add the **memfile**, **mkeep**, **mlock**, and **mmap** directives.
- Add the **mkeep**, **mlock**, and **mmap** attributes to the path directive.

14 March 2005

- Remove documentation on local redirection mode.

1 June 2005

- Describe conditional directives.
- Deprecate the **–r**, **–t**, **–y** command line options.

12 Jan 2006

- Add the **namelib** directive.

22 Mar 2006

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

25 May 2006

- Deprecate the **redirect** directive.
- Document the **authlib** directive.
- Miscellaneous changes to accommodate centralized configuration for clustering.

2 Oct 2006

- Add **sync** and **async** options to **stage** directive.

17 Jan 2007

- Document the **defaults** directive.
- Document the **osslib** directive.
- Deprecate all of the "[no]xxx" directives.

2 Apr 2007

- Deprecate **mssgwcmd** and **path** directives.
- Document the **oss.msscmd** directive (to replace **oss.mssgwcmd**).
- Document the **all.export** directive (to replace **oss.path**).
- Enhance documentation of **oss.defaults**.
- Move conditional directives to a separate manual.
- Document the **stagemsg** directive.

- Document **oss.xfr keep** option.
- Document that the LFN is the primary path name that is externally passed to around.

8 Jan 2008

- Remove documentation on deprecated directives.
- Substitute a generic reference to clustering wherever **olbd** was mentioned.
- General cleanup.

12 Mar 2008

- Document the **usage** directive.

17 Mar 2008

- Document the **notifymsg** directive.
- Document the **xa** option of the **cache** directive.
- Describe extended attributes.

7 Apr 2008

- Update the **ofs.forward** directive. Describe **1way**, **2way**, **3way** modes and the **truncate** operations.
- Add **trunc** to the **notify** and **notifymsg** directives.

2 Jun 2009

- Document **POSC** processing and the **ofs.persist** directive.
- Document the **ofs.posc** OPID.

12 Apr 2010

- Clarify the effects of **check**, **dread**, **rcreate**, and **stage** export attributes on **msscmd** and **stagemcmd** directives.

22 Apr 2010

- Document the fact that the oss now has a built-in interface to the File Residency Manager via **frm_xfrd**.
- Document the **rsscmd** directive as preferred over **msscmd**.
- Document the **space** directive.
- Deprecate the **cache** directive.
- Document that **msscmd**, **rsscmd**, and **stagecmd** no longer have any effect on the default setting of **migrate** and **stage** export options.
- Better explain the side-effects of the **nocheck export** attribute.

- Document the **purge** export option.
- Simplify the **xfr** directive by adding the **deny** option and dropping unused positional parameters.
- Document the **ofs.lcl** opaque ID.
- Remove completely check for undocumented options left secretly behind for backward compatibility (e.g., **mssgwcmd**, **mssgwpath**, **gwbacklog**, all single default options).
- Deprecate the **oss.userprty** directive by allowing the **up** option of **oss.xfr** to do the same thing.

9 Dec 2010

- Correct the **oss.memfile** directive to indicate that it now uses file extended attributes.

27 Dec 2010

- Describe how to setup proxy servers. This includes various directives specific to such a setup as well as tuning options.
- Describe the **netchk** utility.

9 May 2011

- Document the **pss.cache** directive.

31 May 2011

- Document the **ofs.ckslib** and **ofs.cksrdsz** directives.

27 Sep 2011

- Document the **stage+** option on the **oss.defaults** and **all.export** directives.
- Remove documentation on deprecated **oss.cache** and **oss.userprty** directives.
- Remove documentation about out MTS special files now encoded as extended attributes.
- Remove description of **compcheck** and **ssdec defaults/export** attributes, the **compdetect** directive and the **oosquish** command. None of these are used nor fully implemented.
- Add description of the **oss.lcl** opaque identifier.

5 Oct 2011

- Clean-up the proxy section a bit more.
- Describe the **pss.ckslib** and **pss.namelib** directives.

8 Dec 2011

- Document how to specify the cluster management client interface plug-in via the **ofs.cmslib** directive.

---------- **Release 3.1.1**

2 Apr 2012

- Document the **ofs.tpc** directive.

24 Apr 2012

- Document the **pss.origin** directive more explicitly.

20 Jun 2012

- Document the new **oss.xfr fdir** option.

---------- **Release 3.3.6**

15 Nov 2013

- Realign proxy directives and options to correspond to available features in the new client.

8 Apr 2014

- Document the **autorm**, **echo**, and **scan** options of the **ofs.tpc** directive.

---------- **Release 4.0.0**

17 Jun 2014

- Move proxy service documentation to a separate reference.

9 Jul 2014

- Document the **oss.statlib** directive.

24 Oct 2014

- Document the **ofs.xattrlib** directive.
- Document the **+xattr** option in the **ofs.osslib** directive.

12 Nov 2014

- Document the **+cksio** option in the **ofs.osslib** directive.

11 Oct 2015

- Document the **-2** option in the **ofs.statlib** directive.

18 Oct 2015

- Document the **non2n** option in the **ofs.statlib** directive.

27 Oct 2017

- Further enhance the explanation of the **scan** option of **ofs.tpc** directive.

10 Feb 2018

- Document the **-arevents** option in the **ofs.statlib** directive.
- Add leading dash to all **ofs.statlib** directive options for consistency.

23 May 2018

- Document the **xattrs** option in the **oss.defaults** and **all.export** directives.
- Document the **ofs.usxattr** directive.
- Document the **XRDOSSCSCAN** environment variable in the **oss.cachescan** directive.

19 June 2018

- Document the **fcreds** and **fcpath** options in the **ofs.tpc** directive.
- Document the enhanced **streams** option in the **ofs.tpc** directive.
- Add a section to describe using delegated credentials for third party copy.
- Document the equals option in the **ofs.ckslib** directive.

24 August 2018

- Document the **oss.space** directive for assigning spaces to logical file name paths.

1 February 2019

- Document the **ofs.tpc redirect** directive.

2 April 2019

- Document the **ofs.persist sync** option.
- Document the **ofs.preplib** directive.

8 May 2019

- Add a section better explaining the **TPC** script.

15 May 2019

- Document that the **oss.localroot** path must exist and be a directory for default plug-ins.
- Documents the **ofs.xattr** directive.

18 October 2019

- Remove the **+cksio** option from the **ofs.osslib** directive (it's the default).
- Document the **++** option on the **ofs.authlib**, **ofs.osslib**, **ofs.preplib**, and **oss.xattrlib** directives.

22 November 2019

- Provide an example on how the **oss.space** directive can be used.

17 March 2020

- Document the **ofs.ctllib** directive.

20 April 2020

- Remove the **oss.statlib** directive **-2** option as it's now automatically handled. The option is still accepted for backward compatibility.

2 June 2020

- Document the **cache** attribute for the **oss.defaults** and **all.export** directives**.**
- Document the **ofs.dirlist** directive.

16 June 2020

- Document the **sync** option for the **oss.usage** directive**.**

23 June 2020

- Document the **oss.spacescan** directive which replaces the **oss.cachescan** directive**.**
- Remove references to the **oss.cachescan** directive as well as the **oss.cache** directive, the latter is no longer supported.

12 October 2020

- Document the **ofs.chkpnt** directive**.**

20 November 2020

- Re-document the **+cksio** option on the **ofs.osslib** directive**.**

5 April 2021

- Document the **++** option on the **ofs.ckslib** directive**.**

25 June 2021

- Document the **chkmount** option on the **oss.space** directive**.**

30 July 2021

- Document **crc32c** as a natively supported checksum.

8 December 2021

- Document the generic prepare plug-in, **libXrdOfsPrepGPI.so.**