



Project no. IST-033576

# XtreemOS

Integrated Project

BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

## Stable Version of XtreemFS and OSS D3.4.6

Due date of deliverable: 30-NOV-2009  
Actual submission date: 26-MAR-2010

*Start date of project: June 1<sup>st</sup> 2006*

*Type: Deliverable  
WP number: WP3.4*

*Responsible institution: ZIB  
Editor & and editor's address: Björn Kolbeck  
Zuse Institute Berlin  
Takustr. 7  
14195 Berlin  
Germany*

Version 1.0 / Last edited by Kim-Thomas Möller / 23-MAR-2010

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
<b>PU</b>	Public	✓
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	

**Revision history:**

<b>Version</b>	<b>Date</b>	<b>Authors</b>	<b>Institution</b>	<b>Section affected, comments</b>
0.1	16.10.09	Björn Kolbeck, Jan Stender	ZIB	Updated version of XtreamFS User Guide
0.2	20.10.09	Marc-Florian Müller	UDUS	Updated version of OSS User Guide
0.3	23.10.09	Jan Stender	ZIB	Partially re-arranged the structure of the XtreamFS User Guide, added an introduction
0.4	10.11.09	Jan Stender	ZIB	Fixed XtreamFS user guide according to the internal reviewers' comments
0.5	18.03.10	Björn Kolbeck	ZIB	Added a list of changes/new features, listed the changes to the user guide in the summary
0.6	23.03.10	Kim-Thomas Möller	UDUS	Added descriptions of changes/new features in OSS to summary and user guide

**Reviewers:**

Mathijs den Burger (VUA), Roman Talyansky (SAP)

**Tasks related to this deliverable:**

<b>Task No.</b>	<b>Task description</b>	<b>Partners involved<sup>°</sup></b>
T3.4.5	Object Sharing Service	UDUS*
T3.4.7	XtreamFS Client-Side Caching	NEC*
T3.4.8	XtreamFS File Replication	ZIB*
T3.4.9	XtreamFS Automatic Replica Management	BSC*
T3.4.10	XtreamFS Testing, Performance, Compatibility and Maintenance	CNR*
T3.4.11	XtreamFS Consistent Snapshots	ZIB*

<sup>°</sup>This task list may not be equivalent to the list of partners contributing as authors to the deliverable

\*Task leader

# Contents

<b>1</b>	<b>Executive Summary</b>	<b>9</b>
<b>2</b>	<b>The XtreamFS User Guide</b>	<b>11</b>
2.1	Changes and new Features . . . . .	11
2.2	Quick Start . . . . .	12
2.3	About XtreamFS . . . . .	1
2.3.1	What is XtreamFS? . . . . .	1
	What makes XtreamFS a distributed file system?	1
	What makes XtreamFS a replicated file system?	1
2.3.2	Is XtreamFS suitable for me? . . . . .	2
	XtreamFS is ... . . . .	2
	XtreamFS is not ... . . . .	3
2.3.3	Core Features . . . . .	3
	Distribution. . . . .	3
	Replication. . . . .	4
	Striping. . . . .	4
	Security. . . . .	4
2.3.4	Architecture . . . . .	5
	XtreamFS Components. . . . .	5
2.4	XtreamFS Services . . . . .	6
2.4.1	Installation . . . . .	6
	Prerequisites . . . . .	7
	Installing from Pre-Packaged Releases . . . . .	7

---

Installing from Sources . . . . .	7
2.4.2 Configuration . . . . .	8
A Word about UUIDs . . . . .	8
Automatic DIR Discovery . . . . .	9
Authentication . . . . .	9
Configuring SSL Support . . . . .	10
Converting PEM files to PKCS#12. . . . .	10
Importing trusted certificates from PEM into a JKS. . . . .	11
Sample Setup. . . . .	11
List of Configuration Options . . . . .	13
admin_password <i>optional</i> . . . . .	13
authentication_provider . . . . .	13
capability_secret . . . . .	14
checksums.enabled . . . . .	14
checksums.algorithm . . . . .	14
database.dir . . . . .	14
database.log . . . . .	15
debug.level <i>optional</i> . . . . .	15
debug.categories <i>optional</i> . . . . .	16
dir_service.host . . . . .	17
dir_service.port . . . . .	17

---

<code>discover</code> <i>optional</i>	17
<code>geographic_coordinates</code>	17
<code>hostname</code> <i>optional</i>	18
<code>http_port</code>	18
<code>listen.address</code> <i>optional</i>	18
<code>listen.port</code>	18
<code>local_clock_renewal</code>	19
<code>no_atime</code>	19
<code>no_fsync</code> <i>optional</i>	19
<code>object_dir</code>	20
<code>osd_check_interval</code>	20
<code>remote_time_sync</code>	20
<code>report_free_space</code>	20
<code>ssl.enabled</code>	21
<code>ssl.service_creds</code>	21
<code>ssl.service_creds.container</code>	21
<code>ssl.service_creds.pw</code>	21

---

	<code>ssl.trusted_certs</code>	22
	<code>ssl.trusted_certs.container</code>	22
	<code>ssl.trusted_certs.pw</code>	22
	<code>uuid</code>	22
2.4.3	Execution and Monitoring	22
	Starting and Stopping the XtremFS services	22
	Web-based Status Page	23
2.4.4	Troubleshooting	24
2.5	XtremFS Client	24
2.5.1	Installation	25
	Prerequisites	25
	Installing from Pre-Packaged Releases	25
	Installing from Sources	25
2.5.2	Volume Management	26
	Creating Volumes	26
	Deleting Volumes	27
	Listing all Volumes	27
2.5.3	Accessing Volumes	28
	Mounting and Un-mounting	28
	Mount Options	29
2.5.4	Troubleshooting	29
2.6	XtremFS Tools	32
2.6.1	Installation	32
	Prerequisites	32
	Installing from Pre-Packaged Releases	32
	Installing from Sources	33
2.6.2	Maintenance Tools	33

---

	MRC Database Conversion . . . . .	33
	Scrubbing and Cleanup . . . . .	34
2.6.3	User Tools . . . . .	35
	Showing XtreamFS-specific File Info . . . . .	35
	Changing Striping Policies . . . . .	36
	Read-Only Replication . . . . .	37
	Automatic On-Close Replication . . . . .	39
	Changing OSD and Replica Selection Policies . . . . .	40
	Setting and Listing Policy Attributes . . . . .	41
2.7	Policies . . . . .	42
2.7.1	Authentication Policies . . . . .	42
	UNIX uid/gid - NullAuthProvider . . . . .	42
	Plain SSL Certificates - SimpleX509AuthProvider . . . . .	42
	XtreamOS Certificates - XOSAAuthProvider . . . . .	43
2.7.2	Authorization Policies . . . . .	43
2.7.3	OSD and Replica Selection Policies . . . . .	44
	Attributes . . . . .	45
	Predefined Policies . . . . .	45
	Filtering Policies . . . . .	45
	Grouping Policies . . . . .	46
	Sorting Policies . . . . .	47
2.7.4	Striping Policies . . . . .	47
2.7.5	Plug-in Policies . . . . .	48
<b>3</b>	<b>The OSS API and User Guide</b>	<b>51</b>
3.1	Overview . . . . .	51
3.2	Changes and New Features . . . . .	51
3.2.1	Object Allocators . . . . .	51
3.2.2	Local Transaction Commits . . . . .	52
3.2.3	Increased Flexibility . . . . .	52
3.2.4	Nameservice, Overlay Network and Configuration . . . . .	53

---

3.3	Installation of OSS . . . . .	53
3.3.1	Installing OSS using the Distribution Packages . . . . .	53
3.3.2	Building and Installing OSS from Source . . . . .	54
	Prerequisites . . . . .	54
	Compilation . . . . .	55
	Installation . . . . .	55
3.3.3	Testing the OSS Installation . . . . .	56
	Simple test of Object Sharing . . . . .	56
	The Raytracer Application . . . . .	56
3.4	Developing Applications using OSS . . . . .	57
3.4.1	Internal Interface of the OSS Library . . . . .	57
3.4.2	Linking against the OSS Library . . . . .	60
<b>A</b>	<b>XtreemFS Appendix</b>	<b>61</b>
A.1	XtreemFS Support . . . . .	61
A.2	XtreemOS Integration . . . . .	61
	A.2.1 XtreemFS Security Preparations . . . . .	61
A.3	XtreemFS Command Line Utilities . . . . .	63
<b>B</b>	<b>OSS Appendix</b>	<b>65</b>
B.1	OSS Configuration Options . . . . .	65
B.1.1	Debugging . . . . .	65
	debug level for whole build process . . . . .	65
	debug glib . . . . .	65
	debug networking . . . . .	65
	Per-file debug levels . . . . .	66
B.1.2	Code generation . . . . .	66
	Processor Architecture . . . . .	66
B.1.3	Library Interface . . . . .	66
	oss_mmap . . . . .	66
	oss_sync/oss_push/oss_pull . . . . .	66



---

	oss_nameservice_get/oss_nameservice_set . . . . .	66
	nameservice consistency . . . . .	67
	miscellaneous debug functions . . . . .	67
	unstable library interface . . . . .	67
	oss_wait . . . . .	67
B.1.4	Communication . . . . .	67
	Overlay Routing . . . . .	67
	Superpeer Network . . . . .	67
B.1.5	Monitoring . . . . .	68
	monitoring . . . . .	68
	log monitor data to file . . . . .	68
	periodic dump . . . . .	68
	short log . . . . .	68
	object_mmap . . . . .	68
	object_alloc . . . . .	68
	object_free . . . . .	69
	read_fault . . . . .	69
	write_fault . . . . .	69
	read_access . . . . .	69
	write_access . . . . .	69
B.1.6	Memory allocator . . . . .	69
	mospace allocation from dlmalloc . . . . .	69
	millipage implementation . . . . .	69
	simple list allocator . . . . .	70
	replica management . . . . .	70
	diff computation and transfer . . . . .	70
B.1.7	Applications . . . . .	70
	build raytracer . . . . .	70
	build wissenheim . . . . .	70
B.1.8	Remote installation . . . . .	70

---

# Chapter 1

## Executive Summary

This document contains an updated version of the two user guides for the XtremFS File System and for the Object Sharing Service (OSS).

The XtremFS user guide describes the release 1.1 of XtremFS. The most important changes and features are: read-only file replication with support for partial replicas; a more efficient, binary protocol which replaced the text-based protocol; a complete client rewrite; a more efficient metadata server that supports asynchronous snapshots and very large volumes. A complete list of new features can be found in section [2.1](#).

The documentation provided in D3.4.4 has been revised and sections have been updated with information on recently developed features. A more detailed and feature-oriented introduction has been added in sec. [2.3](#) which also describes the new features such as read-only replication. Common problems reported by our users have been added to the troubleshooting section for the client ([2.5.4](#)) and the servers ([2.4.4](#)). The section for the XtremFS client ([2.5](#)) has been rewritten for the new client. Information about the read-only replication has been added in sections [2.3](#), [2.6.3](#) and [2.6.3](#). The section on OSD and Replica selection policies ([2.7.3](#)) have been rewritten to reflect the new policy architecture introduced with XtremFS 1.0. In addition, the entire user guide has been updated to reflect new configuration options and command line tool usage.

The OSS user guide describes the release 0.4 of the Object Sharing Service. It documents the internal library interface as well as the installation and usage of OSS.

Compared to OSS release 0.2, which has been described in D3.4.4, a number of features have been added. The most important changes are the implementation of several object allocators for different purposes, local commits

---

of transactions, and increased flexibility with regard to object allocation and transaction execution. Section 3.2 details these new features and further improvements. In Section 3.4.1, the changes in the internal library interface are described. Appendix B.1 documents the configuration options for OSS 0.4. Performance evaluations are not part of this deliverable; they will be presented in D4.2.6.

# Chapter 2

## The XtreamFS User Guide

### 2.1 Changes and new Features

This is a summary of the most important changes and new features since version 0.10 of XtreamFS:

- **Read-only File Replication**

Files can be replicated across multiple OSDs and datacenters. The read-only replication also supports partial replicas which are filled when clients request objects. The client automatically switches between replicas upon OSD failure; this replica-failover is transparent to the application.

- **Completely re-written Client**

The client has been written from scratch and now supports Mac OS X and Windows natively.

- **ONC RPC based Protocol**

XtreamFS now uses a binary protocol based on ONC RPC for all communication. The communication infrastructure of the servers has been rewritten from scratch. This binary protocol is more efficient and replaces the old HTTP/JSON RPCs.

- **Efficient Metadata Management with BabuDB**

The Metadata server (MRC) was rewritten for the new BabuDB storage backend. BabuDB is an LSM-Tree based database developed at ZIB with a design similar to Google BigTables. This new storage backend allows the MRC to support volumes which are larger than the main memory.

- 
- **Full support for POSIX advisory locks**  
POSIX advisory file locks are required by some applications like mail servers and databases to ensure exclusive access to a file or ranges of a file.
  - **On-close Replication**  
With this option, replicas can automatically be created and filled when a files is closed.
  - **Gridmap User Mappings**  
Gridmap files contain mappings from local to global user IDs and are used in Grid middlewares likes Globus and gLite. The XtreamFS client can now use these, in addition to the XtreamOS AMS mapping service, to map user accounts. This features enhances the interoperability between XtreamOS and other grid middlewares.
  - **Windows-support for XtreamFS Servers**  
XtreamFS servers can now be executed on Windows platforms with Java.
  - **Complete Re-design of OSD and Replica Selection Polices**  
For both tasks, OSD and Replica selection, there is only one type of policies now. This simplifies the management overhead for administrators. The new design allows policies to be composed of several simple policies for selecting, grouping and sorting a list of OSDs.
  - **Non-package Installation**  
XtreamFS can now be installed automatically from the sources without using packages.
  - **Automatic DIR Service Discovery**  
XtreamFS MRCs and OSDs can automatically discover a Directory Service (DIR) on the local network to simplify set-up.

## 2.2 Quick Start

This is the very short version to help you set up a local installation of XtreamFS.

1. Download XtreamFS RPMs/DEBs and install

- 
- (a) Download the RPMs or DEBs for your system from the XtreamFS website (<http://www.xtreemfs.org>)
  - (b) open a root console (`su` or `sudo`)
  - (c) install with `rpm -Uhv xtreemfs-client-1.1.x.rpm xtreemfs-server-1.1.x.rpm`
2. Start the Directory Service:  
`/etc/init.d/xtreemfs-dir start`
  3. Start the Metadata Server:  
`/etc/init.d/xtreemfs-mrc start`
  4. Start the OSD:  
`/etc/init.d/xtreemfs-osd start`
  5. If not already loaded, load the FUSE kernel module:  
`modprobe fuse`
  6. Depending on your distribution, you may have to add users to a special group to allow them to mount FUSE file systems. In openSUSE users must be in the group `trusted`, in Ubuntu in the group `fuse`. You may need to log out and log in again for the new group membership to become effective.
  7. You can now close the root console and work as a regular user.
  8. Wait a few seconds for the services to register at the directory service. You can check the registry by opening the DIR status page in your favorite web browser <http://localhost:30638>.
  9. Create a new volume with the default settings:  
`xtfs_mkvol localhost/myVolume`
  10. Create a mount point:  
`mkdir ~/xtreemfs`
  11. Mount XtreamFS on your computer:  
  
`xtfs_mount localhost/myVolume ~/xtreemfs`
  12. Have fun ;-)
  13. To un-mount XtreamFS:  
`xtfs_umount ~/xtreemfs`

---

You can also mount this volume on remote computers. First make sure that the ports 32636, 32638 and 32640 are open for incoming TCP connections. You must also specify a hostname that can be resolved by the remote machine! This hostname has to be used instead of `localhost` when mounting.

## 2.3 About XtreamFS

Since you decided to take a look at this user guide, you probably read or heard about XtreamFS and want to find out more. This chapter contains basic information about the characteristics and the architecture of XtreamFS.

### 2.3.1 What is XtreamFS?

XtreamFS is a file system for a variety of different use cases and purposes. Since it is impossible to categorize or explain XtreamFS in a single sentence, we introduce XtreamFS by means of its two most significant properties: *XtreamFS is a globally distributed and replicated file system.*

**What makes XtreamFS a distributed file system?** We consider a file system as *distributed* if files are stored across a number of servers rather than a single server or local machine. Unlike local or network file systems, a distributed file system aggregates the capacity of multiple servers. As a *globally distributed* file system, XtreamFS servers may be dispersed all over the world. The capacity can be increased and decreased by adding and removing servers, but from a user's perspective, the file system appears to reside on a single machine.

**What makes XtreamFS a replicated file system?** We call it a *replicated* file system because replication is one of its most prominent features. XtreamFS is capable of maintaining replicas of files on different servers. Thus, files remain accessible even if single servers, hard disks or network connections fail. Besides, replication yields benefits in terms of data rates and access times. Different replicas of a file can be accessed simultaneously on different servers, which may lead to a better performance compared to simultaneous accesses on a single server. By placing file replicas close the consuming users and applications in a globally distributed installation, the effects of network latency and bandwidth reduction in wide area networks can be mitigated. However, replication is transparent to users and applications that work with



---

XtreemFS; the file system is capable of controlling the life cycle and access of replicas without the need for human intervention or modifications of existing applications.

### 2.3.2 Is XtreemFS suitable for me?

If you consider using XtreemFS, you may be a system administrator in search of a better and more flexible alternative to your current data management solution. Or you may be a private user in need of a file system that can be easily set up and accessed from any machine in the world. You might also be someone looking for an open-source solution to manage large amounts of data distributed across multiple sites. In any case, you will wonder if XtreemFS fulfills your requirements. As a basis for your decision, the following two paragraphs point out the characteristics of XtreemFS.

#### **XtreemFS is ...**

- ... an open source file system. It is distributed freely and can be used by anyone without limitations.
- ... a POSIX file system. Users can mount and access XtreemFS like any other common file system. Application can access XtreemFS via the standard file system interface, i.e. without having to be rebuilt against a specialized API. XtreemFS supports a POSIX-compliant access control model.
- ... a multi-platform file system. Server and client modules can be installed and run on different platforms, including most Linux distributions, Solaris, Mac OS X and Windows.
- ... a globally distributed file system. Unlike cluster file systems, an XtreemFS installation is not restricted to a single administrative domain or cluster. It can span the globe and may comprise servers in different administrative domains.
- ... a failure-tolerant file system. As stated in the previous section, replication can keep the system alive and the data safe. In this respect, XtreemFS differs from most other open-source file systems.
- ... a secure file system. To ensure security in an untrusted, worldwide network, all network traffic can be encrypted with SSL connections, and users can be authenticated with X.509 certificates.

---

... a customizable file system. Since XtreamFS can be used in different environments, we consider it necessary to give administrators the possibility of adapting XtreamFS to the specific needs of their users. Customizable policies make it possible change the behavior of XtreamFS in terms of authentication, access control, striping, replica placement, replica selection and others. Such policies can be selected from a set of predefined policies, or implemented by administrators and plugged in the system.

### **XtreamFS is not ...**

... a high-performance cluster file system. Even though XtreamFS reaches acceptable throughput rates on a local cluster, it cannot compete with specialized cluster file systems in terms of raw performance numbers. Most such file systems have an optimized network stack and protocols, and a substantially larger development team. If you have huge amounts of data on a local cluster with little requirements but high throughput rates to them, a cluster file system is probably the better alternative.

... a replacement for a local file system. Even though XtreamFS can be set up and mounted on a single machine, the additional software stack degrades the performance, which makes XtreamFS a bad alternative.

### **2.3.3 Core Features**

The core functionality of XtreamFS is characterized by a small set of features, which are explained in the following.

**Distribution.** An XtreamFS installation comprises multiple servers that may run on different nodes connected on a local cluster or via the Internet. Provided that the servers are reachable, a client module installed on any machine in the world can access the installation. A binary communication protocol based on Sun's ONC-RPC ensures an efficient communication with little overhead between clients and servers. XtreamFS ensures that the file system remains in a consistent state even if multiple clients access a common set of files and directories. Similar to NFS, it offers a close-to-open consistency model in the event of concurrent file accesses.

---

**Replication.** Since version 1.0, XtremFS supports *read-only replication*. A file may have multiple replicas, provided that it was explicitly made read-only before, which means that its content cannot be changed anymore. This kind of replication can be used to make write-once files available to many consumers, or to protect them from losses due to hardware failures. Besides complete replicas that are immediately synchronized after having been created, XtremFS also supports partial replicas that are only filled with content on demand. They can e.g. be used to make large files accessible to many clients, of which only parts need to be accessed.

Currently, XtremFS does not support replication of mutable files. From a technical perspective, this is more challenging than read-only replication, since XtremFS has to ensure that all replicas of a file remain consistent despite attempts to concurrently write different replicas, as well as network and component failures. However, we are planning on supporting full read-write replication with future XtremFS releases.

**Striping.** To ensure acceptable I/O throughput rates when accessing large files, XtremFS supports *striping*. A striped file is split into multiple chunks (“*stripes*”), which are stored on different storage servers. Since different stripes can be accessed in parallel, the whole file can be read or written with the aggregated network and storage bandwidth of multiple servers. XtremFS currently supports the RAID0 striping pattern, which splits a file up in a set of stripes of a fixed size, and distributes them across a set of storage servers in a round-robin fashion. The size of an individual stripe as well as the number of storage servers used can be configured on a per-file or per-directory basis.

**Security.** To enforce security, XtremFS offers mechanisms for user authentication and authorization, as well as the possibility to encrypt network traffic.

*Authentication* describes the process of verifying a user’s or client’s identity. By default, authentication in XtremFS is based on local user names and depends on the trustworthiness of clients and networks. In case a more secure solution is needed, X.509 certificates can be used.

*Authorization* describes the process of checking user permissions to execute an operation. XtremFS supports the standard UNIX permission model, which allows for assigning individual access rights to file owners, owning groups and other users.

---

Authentication and authorization are policy-based, which means that different models and mechanisms can be used to authenticate and authorize users. Besides, the policies are pluggable, i.e. they can be freely defined and easily extended.

XtreemFS uses unauthenticated and unencrypted TCP connections by default. To encrypt all network traffic, services and clients can establish *SSL* connections. However, using SSL requires that all users and services have valid X.509 certificates.

### 2.3.4 Architecture

XtreemFS implements an *object-based file system architecture* (Fig. 2.1): file content is split into a series of fixed-size *objects* and stored across storage servers, while *metadata* is stored on a separate metadata server. The metadata server organizes file system metadata as a set of *volumes*, each of which implements a separate file system namespace in form of a directory tree.

In contrast to block-based file systems, the management of available and used storage space is offloaded from the metadata server to the storage servers. Rather than inode lists with block addresses, file metadata contains lists of storage servers responsible for the objects, together with striping policies that define how to translate between byte offsets and object IDs. This implies that object sizes may vary from file to file.

**XtreemFS Components.** An XtreemFS installation contains three types of servers that can run on one or several machines (Fig. 2.1):

- DIR - Directory Service  
The directory service is the central registry for all services in XtreemFS. The MRC uses it to discover storage servers.
- MRC - Metadata and Replica Catalog  
The MRC stores the directory tree and file metadata such as file name, size or modification time. Moreover, the MRC authenticates users and authorizes access to files.
- OSD - Object Storage Device  
An OSD stores arbitrary objects of files; clients read and write file data on OSDs.

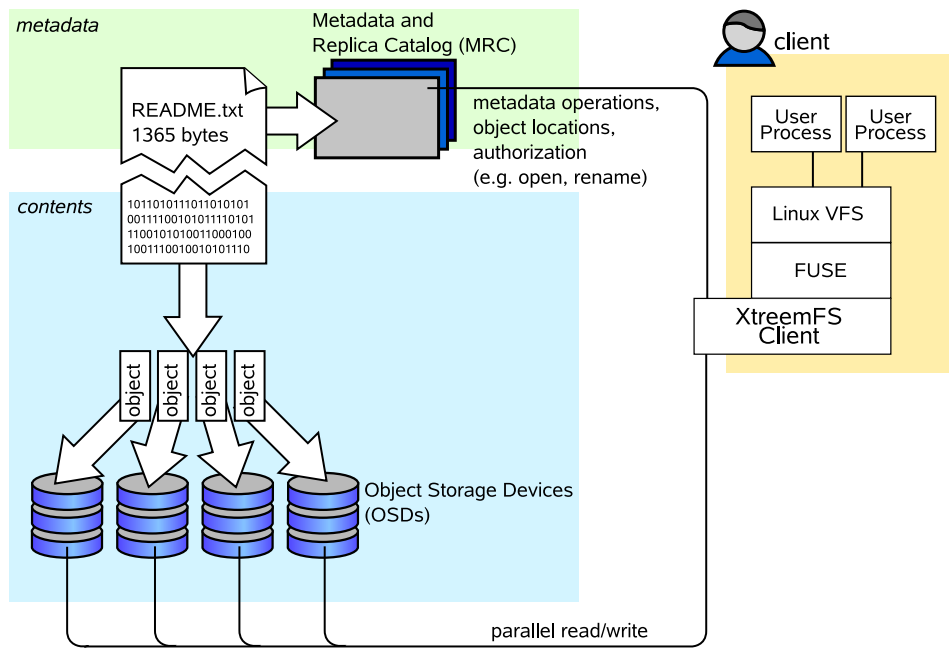


Figure 2.1: The XtremFS architecture and components.

These servers are connected by the *client* to a file system. A client *mounts* one of the volumes of the MRC in a local directory. It translates file system calls into RPCs sent to the respective servers.

The client is implemented as a *FUSE user-level driver* that runs as a normal process. FUSE itself is a kernel-userland hybrid that connects the user-land driver to Linux' *Virtual File System (VFS)* layer where file system drivers usually live.

## 2.4 XtremFS Services

This chapter describes how to install and set up the server side of an XtremFS installation.

### 2.4.1 Installation

When installing XtremFS server components, you can choose from two different installation sources: you can download one of the *pre-packaged releases* that we create for most Linux distributions or you can install directly from the *source tarball*.

---

Note that the source tarball contains the complete distribution of XtreamFS, which also includes client and tools. Currently, binary distributions of the server are only available for Linux.

## Prerequisites

For the pre-packaged release, you will need Sun Java JRE 1.6.0 or newer to be installed on the system.

When building XtreamFS directly from the source, you need a Sun Java JDK 1.6.0 or newer, Ant 1.6.5 or newer and gmake.

## Installing from Pre-Packaged Releases

On RPM-based distributions (RedHat, Fedora, SuSE, Mandriva, XtreamOS) you can install the package with

```
$> rpm -i xtreamfs-server-1.1.x.rpm
```

For Debian-based distributions, please use the `.deb` package provided and install it with

```
$> dpkg -i xtreamfs-server-1.1.x.deb
```

Both packages will also install `init.d` scripts for an automatic start-up of the services. Use `insserv xtreamfs-dir`, `insserv xtreamfs-mrc` and `insserv xtreamfs-osd`, respectively, to automatically start the services during boot.

## Installing from Sources

Extract the tarball with the sources. Change to the top level directory and execute

```
$> make server
```

This will build the XtreamFS server and Java-based tools. When done, execute

```
$> sudo make install
```

---

to install the server components. Finally, you will be asked to execute a post-installation script

```
$> sudo /etc/xos/xtreemfs/postinstall_setup.sh
```

to complete the installation.

## 2.4.2 Configuration

After having installed the XtreamFS server components, it is recommendable to configure the different services. This section describes the different configuration options.

XtreamFS services are configured via Java properties files that can be modified with a normal text editor. Default configuration files for a Directory Service, MRC and OSD are located in `/etc/xos/xtreemfs/`.

### A Word about UUIDs

XtreamFS uses UUIDs (Universally Unique Identifiers) to be able to identify services and their associated state independently from the machine they are installed on. This implies that you cannot change the UUID of an MRC or OSD after it has been used for the first time!

The Directory Service resolves UUIDs to service endpoints, where each service endpoint consists of an IP address or hostname and port number. Each endpoint is associated with a netmask that indicates the subnet in which the mapping is valid. In theory, multiple endpoints can be assigned to a single UUID if endpoints are associated with different netmasks. However, it is currently only possible to assign a single endpoint to each UUID; the netmask must be “\*”, which means that the mapping is valid in all networks. Upon first start-up, OSDs and MRCs will auto-generate the mapping if it does not exist, by using the first available network device with a public address.

Changing the IP address, hostname or port is possible at any time. Due to the caching of UUIDs in all components, it can take some time until the new UUID mapping is used by all OSDs, MRCs and clients. The TTL (time-to-live) of a mapping defines how long an XtreamFS component is allowed to keep entries cached. The default value is 3600 seconds (1 hour). It should be set to shorter durations if services change their IP address frequently.

---

To create a globally unique UUID you can use tools like `uuidgen`. During installation, the post-install script will automatically create a UUID for each OSD and MRC if it does not have a UUID assigned.

## Automatic DIR Discovery

OSDs and MRCs are capable of automatically discovering a Directory Service. If automatic DIR discovery is switched on, the service will broadcast requests to the local LAN and wait up to 10s for a response from a DIR. The services will select the first DIR which responded, which can lead to non-deterministic behavior if multiple DIR services are present. Note that the feature works only in a local LAN environment, as broadcast messages are not routed to other networks. Local firewalls on the computers on which the services are running can also prevent the automatic discovery from working.

**Security:** The automatic discovery is a potential security risk when used in untrusted environments as any user can start-up DIR services.

A statically configured DIR address and port can be used to disable DIR discovery in the OSD and MRC (see Sec. 2.4.2, `dir_service`). By default, the DIR responds to UDP broadcasts. To disable this feature, set `discover = false` in the DIR service config file.

## Authentication

Administrators may choose the way of authenticating users in XtremFS. *Authentication Providers* are pluggable modules that determine how users are authenticated. For further details, see Sec. 2.7.1.

To set the authentication provider, it is necessary to set the following property in the MRC configuration file:

```
authentication_provider = <classname>
```

By default, the following class names can be used:

- `org.xtreemfs.common.auth.NullAuthProvider`  
uses local user and group IDs
- `org.xtreemfs.common.auth.SimpleX509AuthProvider`  
uses X.509 certificates; user and group IDs are extracted from the distinguished names of the certificates



- 
- `org.xtreemos.XtreemOSAuthProvider`  
uses XOSCertificates

## Configuring SSL Support

In order to enable certificate-based authentication in an XtremFS installation, services need to be equipped with X.509 certificates. Certificates are used to establish a mutual trust relationship among XtremFS services and between the XtremFS client and XtremFS services.

Note that it is not possible to mix SSL-enabled and non-SSL services in an XtremFS installation!

Each XtremFS service needs a certificate and a private key in order to be run. Once they have been created and signed, the credentials may need to be converted into the correct file format. XtremFS services also need a *trust store* that contains all trusted Certification Authority certificates.

By default, certificates and credentials for XtremFS services are stored in

```
/etc/xos/xtreemfs/truststore/certs
```

**Converting PEM files to PKCS#12.** The simplest way to provide the credentials to the services is by converting your signed certificate and private key into a PKCS#12 file using `openssl`:

```
$> openssl pkcs12 -export -in ds.pem -inkey ds.key \  
-out ds.p12 -name "DS"  
$> openssl pkcs12 -export -in mrc.pem -inkey mrc.key \  
-out mrc.p12 -name "MRC"  
$> openssl pkcs12 -export -in osd.pem -inkey osd.key \  
-out osd.p12 -name "OSD"
```

This will create three PKCS12 files (`ds.p12`, `mrc.p12` and `osd.p12`), each containing the private key and certificate for the respective service. The passwords chosen when asked must be set as a property in the corresponding service configuration file.

---

**Importing trusted certificates from PEM into a JKS.** The certificate (or multiple certificates) from your CA (or CAs) can be imported into a Java Keystore (JKS) using the Java keytool which comes with the Java JDK or JRE.

Execute the following steps for each CA certificate using the same keystore file.

```
$> keytool -import -alias rootca -keystore trusted.jks \  
-trustcacerts -file ca-cert.pem
```

This will create a new Java Keystore `trusted.jks` with the CA certificate in the current working directory. The password chosen when asked must be set as a property in the service configuration files.

Note: If you get the following error

```
keytool error: java.lang.Exception: Input not an X.509 certificate
```

you should remove any text from the beginning of the certificate (until the -----BEGIN CERTIFICATE----- line).

**Sample Setup.** Users can easily set up their own CA (certificate authority) and create and sign certificates using `openssl` for a test setup.

1. Set up your test CA.

- (a) Create a directory for your CA files

```
$> mkdir ca
```

- (b) Create a private key and certificate request for your CA.

```
$> openssl req -new -newkey rsa:1024 -nodes -out ca/ca.csr \  
-keyout ca/ca.key
```

Enter something like XtreamFS-DEMO-CA as the common name (or something else, but make sure the name is different from the server and client name!).

- (c) Create a self-signed certificate for your CA which is valid for one year.

```
$> openssl x509 -trustout -signkey ca/ca.key -days 365 -req \  
-in ca/ca.csr -out ca/ca.pem
```

- 
- (d) Create a file with the CA's serial number
- ```
$> echo "02" > ca/ca.srl
```
2. Set up the certificates for the services and the XtremFS Client.  
Replace *service* with *dir*, *mrc*, *osd* and *client*.
- (a) Create a private key for the service.  
Use *XtremFS-DEMO-service* as the common name for the certificate.
- ```
$> openssl req -new -newkey rsa:1024 -nodes  
-out service.req  
-keyout service.key
```
- (b) Sign the certificate with your demo CA.  
The certificate is valid for one year.
- ```
$> openssl x509 -CA ca/ca.pem -CAkey ca/ca.key  
-CAserial ca/ca.srl -req  
-in service.req  
-out service.pem -days 365
```
- (c) Export the service credentials (certificate and private key) as a PKCS#12 file.  
Use "passphrase" as export password. You can leave the export password empty for the XtremFS Client to avoid being asked for the password on mount.
- ```
$> openssl pkcs12 -export -in service.pem -inkey service.key  
-out service.p12 -name "service"
```
- (d) Copy the PKCS#12 file to the certificates directory.
- ```
$> mkdir -p /etc/xos/xtreemfs/truststore/certs  
$> cp service.p12 /etc/xos/xtreemfs/truststore/certs
```
3. Export your CA's certificate to the trust store and copy it to the certificate dir.  
You should answer "yes" when asked "Trust this certificate".  
Use "passphrase" as passphrase for the keystore.
- ```
$> keytool -import -alias ca -keystore trusted.jks \  
-trustcacerts -file ca/ca.pem  
$> cp trusted.jks /etc/xos/xtreemfs/truststore/certs
```

- 
4. Configure the services. Edit the configuration file for all your services. Set the following configuration options (see Sec. 2.4.2 for details).

```
ssl.enabled = true
ssl.service_creds.pw = passphrase
ssl.service_creds.container = pkcs12
ssl.service_creds = /etc/xos/xtreemfs/truststore/certs/service.p12
ssl.trusted_certs = /etc/xos/xtreemfs/truststore/certs/trusted.jks
ssl.trusted_certs.pw = passphrase
ssl.trusted_certs.container = jks
```

5. Start up the XtremFS services (see Sec. 2.4.3).

6. Create a new volume (see Sec. 2.5.2 for details).

```
$> xtfs_mkvol --pkcs12-file-path=\
/etc/xos/xtreemfs/truststore/certs/client.p12 localhost/test
```

7. Mount the volume (see Sec. 2.5.3 for details).

```
$> xtfs_mount --pkcs12-file-path=\
/etc/xos/xtreemfs/truststore/certs/client.p12 localhost/test /mnt
```

## List of Configuration Options

All configuration parameters that may be used to define the behavior of the different services are listed in this section. Unless marked as optional, a parameter has to occur (exactly once) in a configuration file.

### `admin_password` *optional*

Services DIR, MRC, OSD

Values String

Default

Description Defines the admin password that must be sent to authorize requests like volume creation, deletion or shutdown.

### `authentication_provider`

Services MRC

Values Java class name

Default `org.xtreemfs.common.auth.NullAuthProvider`

Description Defines the Authentication Provider to use to retrieve the user identity (user ID and group IDs). See Sec. 2.4.2 for details.

---

**capability\_secret**

Services MRC, OSD  
Values String  
Default  
Description Defines a shared secret between the MRC and all OSDs. The secret is used by the MRC to sign capabilities, i.e. security tokens for data access at OSDs. In turn, an OSD uses the secret to verify that the capability has been issued by the MRC.

**checksums.enabled**

Services OSD  
Values true, false  
Default false  
Description If set to true, the OSD will calculate and store checksums for newly created objects. Each time a checksummed object is read, the checksum will be verified.

**checksums.algorithm**

Services OSD  
Values Adler32, CRC32  
Default Adler32  
Description Must be specified if `checksums.enabled` is enabled. This property defines the algorithm used to create OSD checksums.

**database.dir**

Services DIR, MRC  
Values absolute file system path to a directory  
Default DIR: `/var/lib/xtreemfs/dir/database`,  
MRC: `/var/lib/xtreemfs/mrc/database`  
Description The directory in which the Directory Service or MRC will store their databases. This directory should never be on the same partition as any OSD data, if both services reside on the same machine. Otherwise, deadlocks may occur if the partition runs out of free disk space!

---

**database.log**

Services	MRC
Values	absolute file system path
Default	MRC: <code>/var/lib/xtreemfs/mrc/dblog</code>
Description	The directory the MRC uses to store database logs. This directory should never be on the same partition as any OSD data, if both services reside on the same machine. Otherwise, deadlocks may occur if the partition runs out of free disk space!

**debug.level** *optional*

Services	DIR, MRC, OSD
Values	0, 1, 2, 3, 4, 5, 6, 7
Default	6
Description	The debug level determines the amount and detail of information written to logfiles. Any debug level includes log messages from lower debug levels. The following log levels exist:

0 - fatal errors

1 - alert messages

2 - critical errors

3 - normal errors

4 - warnings

5 - notices

6 - info messages

7 - debug messages

---

`debug.categories` *optional*

Services DIR, MRC, OSD  
Values all, lifecycle, net, auth, stage, proc, db, misc  
Default all  
Description Debug categories determine the domains for which log messages will be printed. By default, there are no domain restrictions, i.e. log messages from all domains will be included in the log. The following categories can be selected:

all - no restrictions on the category

lifecycle - service lifecycle-related messages, including startup and shutdown events

net - messages pertaining to network traffic and communication between services

auth - authentication and authorization-related messages

stage - messages pertaining to the flow of requests through the different stages of a service

proc - messages about the processing of requests

db - messages that are logged in connection with database accesses

misc - any other log messages that do not fit in one of the previous categories

Note that it is possible to specify multiple categories by means of a comma or space-separated list.

---

`dir_service.host`

Services	MRC, OSD
Values	hostname or IP address
Default	localhost
Description	Specifies the hostname or IP address of the directory service (DIR) at which the MRC or OSD should register. The MRC also uses this Directory Service to find OSDs. If set to <code>.autodiscover</code> the service will use the automatic DIR discovery mechanism (see Sec. <a href="#">2.4.2</a> ). (Note that the initial ‘.’ is used to avoid ambiguities with hosts called “autodiscover”.)

`dir_service.port`

Services	MRC, OSD
Values	1 .. 65535
Default	32638
Description	Specifies the port on which the remote directory service is listening. Must be identical to the <code>listen_port</code> in your directory service configuration.

`discover` *optional*

Services	DIR
Values	true, false
Default	true
Description	If set to true the DIR will received UDP broadcasts and advertise itself in response to XtreamFS components using the DIR automatic discovery mechanism. If set to false, the DIR will ignore all UDP traffic. For details see Sec. <a href="#">2.4.2</a> .

`geographic_coordinates`

Services	DIR, MRC, OSD
Values	String
Default	
Description	Specifies the geographic coordinates which are registered with the directory service. Used e.g. by the web console.



---

**hostname** *optional*

Services MRC, OSD  
Values String  
Default  
Description If specified, it defines the host name that is used to register the service at the directory service. If not specified, the host address defined in `listen.address` will be used if specified. If neither `hostname` nor `listen.address` are specified, the service itself will search for externally reachable network interfaces and advertise their addresses.

**http\_port**

Services DIR, MRC, OSD  
Values 1 .. 65535  
Default 30636 (MRC), 30638 (DIR), 30640 (OSD)  
Description Specifies the listen port for the HTTP service that returns the status page.

**listen.address** *optional*

Services OSD  
Values IP address  
Default  
Description If specified, it defines the interface to listen on. If not specified, the service will listen on all interfaces (any).

**listen.port**

Services DIR, MRC, OSD  
Values 1 .. 65535  
Default DIR: 32638,  
MRC: 32636,  
OSD: 32640  
Description The port to listen on for incoming ONC-RPC connections (TCP). The OSD uses the specified port for both TCP and UDP. Please make sure to configure your firewall to allow incoming TCP traffic (plus UDP traffic, in case of an OSD) on the specified port.

---

`local_clock_renewal`

Services	MRC, OSD
Values	milliseconds
Default	50
Description	Reading the system clock is a slow operation on some systems (e.g. Linux) as it is a system call. To increase performance, XtremFS services use a local variable which is only updated every <code>local_clock_renewal</code> milliseconds.

`no_atime`

Services	MRC
Values	true, false
Default	true
Description	The POSIX standard defines that the <code>atime</code> (timestamp of last file access) is updated each time a file is opened, even for read. This means that there is a write to the database and hard disk on the MRC each time a file is read. To reduce the load, many file systems (e.g. <code>ext3</code> ) including XtremFS can be configured to skip those updates for performance. It is strongly suggested to disable <code>atime</code> updates by setting this parameter to true.

`no_fsync` *optional*

Services	MRC
Values	true, false
Default	false
Description	By default, the MRC will write all file-modifying operations (such as create file, delete etc.) to disk followed by a <code>fsync</code> to ensure data is written to the hard disk. While this ensures maximum data safety in case of crash of the MRC server, it also reduces the performance of the MRC. Set this to true, if you want much higher performance at the risk of losing uncommitted file operations in case of a server crash.

---

### `object_dir`

Services OSD  
Values absolute file system path to a directory  
Default `/var/lib/xtreemfs/osd/`  
Description The directory in which the OSD stores the objects. This directory should never be on the same partition as any DIR or MRC database, if both services reside on the same machine. Otherwise, deadlocks may occur if the partition runs out of free disk space!

### `osd_check_interval`

Services MRC  
Values seconds  
Default 300  
Description The MRC regularly asks the directory service for suitable OSDs to store files on (see OSD Selection Policy, Sec. 2.7.3). This parameter defines the interval between two updates of the list of suitable OSDs.

### `remote_time_sync`

Services MRC, OSD  
Values milliseconds  
Default 30,000  
Description MRCs and OSDs all synchronize their clocks with the directory service to ensure a loose clock synchronization of all services. This is required for leases to work correctly. This parameter defines the interval in milliseconds between time updates from the directory service.

### `report_free_space`

Services OSD  
Values true, false  
Default true  
Description If set to true, the OSD will report its free space to the directory service. Otherwise, it will report zero, which will cause the OSD not to be used by the OSD Selection Policies (see Sec. 2.7.3).

---

**ssl.enabled**  
Services DIR, MRC, OSD  
Values true, false  
Default false  
Description If set to true, the service will use SSL to authenticate and encrypt connections. The service will not accept non-SSL connections if `ssl.enabled` is set to true.

**ssl.service\_creds**  
Services DIR, MRC, OSD  
Values path to file  
Default DIR: `/etc/xos/xtreemfs/truststore/certs/ds.p12`,  
MRC: `/etc/xos/xtreemfs/truststore/certs/mrc.p12`,  
OSD: `/etc/xos/xtreemfs/truststore/certs/osd.p12`  
Description Must be specified if `ssl.enabled` is enabled. Specifies the file containing the service credentials (X.509 certificate and private key). PKCS#12 and JKS format can be used, set `ssl.service_creds.container` accordingly. This file is used during the SSL handshake to authenticate the service.

**ssl.service\_creds.container**  
Services DIR, MRC, OSD  
Values pkcs12 or JKS  
Default pkcs12  
Description Must be specified if `ssl.enabled` is enabled. Specifies the file format of the `ssl.service_creds` file.

**ssl.service\_creds.pw**  
Services DIR, MRC, OSD  
Values String  
Default  
Description Must be specified if `ssl.enabled` is enabled. Specifies the password which protects the credentials file `ssl.service_creds`.

---

**ssl.trusted\_certs**  
Services DIR, MRC, OSD  
Values path to file  
Default /etc/xos/xtreemfs/truststore/certs/xosrootca.jks  
Description Must be specified if `ssl.enabled` is enabled. Specifies the file containing the trusted root certificates (e.g. CA certificates) used to authenticate clients.

**ssl.trusted\_certs.container**  
Services DIR, MRC, OSD  
Values pkcs12 or JKS  
Default JKS  
Description Must be specified if `ssl.enabled` is enabled. Specifies the file format of the `ssl.trusted_certs` file.

**ssl.trusted\_certs.pw**  
Services DIR, MRC, OSD  
Values String  
Default  
Description Must be specified if `ssl.enabled` is enabled. Specifies the password which protects the trusted certificates file `ssl.trusted_certs`.

**uuid**  
Services MRC, OSD  
Values String, but limited to alphanumeric characters, - and .  
Default  
Description Must be set to a unique identifier, preferably a UUID according to RFC 4122. UUIDs can be generated with `uuidgen`. Example: `eacb6bab-f444-4ebf-a06a-3f72d7465e40`.

### 2.4.3 Execution and Monitoring

This section describes how to execute and monitor XtremFS services.

#### Starting and Stopping the XtremFS services

If you installed a *pre-packaged release* you can start, stop and restart the services with the `init.d` scripts:

```

$> /etc/init.d/xtreemfs-ds start
$> /etc/init.d/xtreemfs-mrc start
$> /etc/init.d/xtreemfs-osd start

```

or

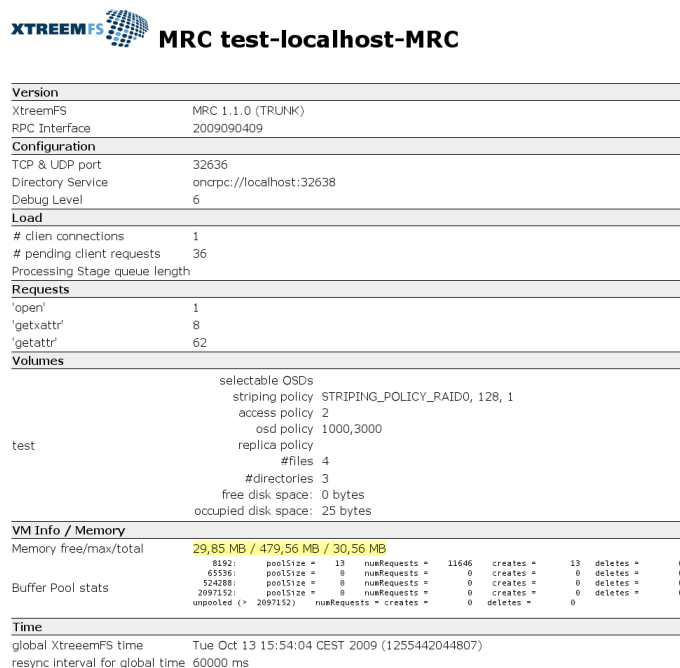
```


$> /etc/init.d/xtreemfs-ds stop
$> /etc/init.d/xtreemfs-mrc stop
$> /etc/init.d/xtreemfs-osd stop

```

To run init.d scripts, root permissions are required. **Note** that the Directory Service must be started first, since a running Directory Service is required when starting an MRC or OSD. Once a Directory Service as well as at least one OSD and MRC are running, XtremFS is operational.

## Web-based Status Page



**XTREEMFS**  **MRC test-localhost-MRC**

Version	
XtremFS	MRC 1.1.0 (TRUNK)
RPC Interface	2009090409
Configuration	
TCP & UDP port	32636
Directory Service	oncrpc://localhost:32638
Debug Level	6
Load	
# clien connections	1
# pending client requests	36
Processing Stage queue length	
Requests	
'open'	1
'getattr'	8
'getattr'	62
Volumes	
	selectable OSDs
	striping policy STRIPING_POLICY_RAID0, 128, 1
	access policy 2
	osd policy 1000,3000
test	replica policy
	#files 4
	#directories 3
	free disk space: 0 bytes
	occupied disk space: 25 bytes
VM Info / Memory	
Memory free/max/total	29,85 MB / 479,56 MB / 30,56 MB
Buffer Pool stats	6192: poolSize = 13 numRequests = 11646 creates = 13 deletes = 0 65536: poolSize = 0 numRequests = 0 creates = 0 deletes = 0 524288: poolSize = 0 numRequests = 0 creates = 0 deletes = 0 2097152: poolSize = 0 numRequests = 0 creates = 0 deletes = 0 unpooled (> 2097152) numRequests = creates = 0 deletes = 0
Time	
global XtremFS time	Tue Oct 13 15:54:04 CEST 2009 (1255442044807)
resync interval for global time	60000 ms

Figure 2.2: OSD status web page

Each XtremFS service can generate an HTML status page, which displays runtime information about the service (Fig. 2.2). The HTTP server that generates the status page runs on the port defined by the configuration property

---

`http_port`; default values are 30636 for MRCs, 30638 for Directory Services, and 30640 for OSDs.

The status page of an MRC can e.g. be shown by opening

`http://my-mrc-host.com:30636/`

with a common web browser.

## 2.4.4 Troubleshooting

Various issues may occur when attempting to set up an XtremFS server component. If a service fails to start, the log file often reveals useful information. Server log files are located in `/var/log/xtremfs`. Note that you can restrict granularity and categories of log messages via the configuration properties `debug.level` and `debug.categories` (see Sec. 2.4.2).

If an error occurs, please check if one of the following requirements is not met:

- You have root permissions when starting the service. Running the `init.d` scripts requires root permissions. However, the services themselves are started on behalf of a user `xtremfs`.
- DIR has been started before MRC and OSD. Problems may occur if a script starts multiple services as background processes.
- There are no firewall restrictions that keep XtremFS services from communicating with each other. The default ports that need to be open are: 32636 (MRC, TCP), 32638 (DIR, TCP), and 32640 (OSD, TCP & UDP).
- The MRC database version is correct. In case of an outdated database version, the `xtfs_mrcdbtool` commands of the old and new XtremFS version can dump and restore the database, respectively (see Sec. 2.6.2).
- A network interface is available on the host. It may be either bound to an IPv4 or IPv6 address.

## 2.5 XtremFS Client

The XtremFS client is needed to access an XtremFS installation from a remote machine. This chapter describes how to use the XtremFS client in order to work with XtremFS like a local file system.

---

## 2.5.1 Installation

There are two different installation sources for the XtreamFS Client: *pre-packaged releases* and *source tarballs*.

Note that the source tarball contains the complete distribution of XtreamFS, which also includes server and tools. Currently, binary distributions of the client are only available for Linux and Windows.

### Prerequisites

To install XtreamFS on Linux, please make sure that FUSE 2.6 or newer, openssl 0.9.8 or newer and a Linux 2.6 kernel are available on your system. For an optimal performance, we suggest to use FUSE 2.8 with a kernel version 2.6.26 or newer.

To build the Linux XtreamFS Client from the source distribution, you also need the openssl headers (e.g. openssl-devel package), python  $\geq$  2.4, and gcc-c++  $\geq$  4.2.

### Installing from Pre-Packaged Releases

On RPM-based distributions (RedHat, Fedora, SuSE, Mandriva, XtreamOS) you can install the package with

```
$> rpm -i xtreamfs-client-1.1.x.rpm
```

For Debian-based distributions, please use the `.deb` package provided and install it with

```
$> dpkg -i xtreamfs-client-1.1.x.deb
```

For Windows, please use the `.msi` installer that will guide you through the installation process.

### Installing from Sources

Extract the tarball with the sources. Change to the top level directory and execute



---

```
$> make client
```

This will build the XtreamFS client and non-Java-based tools. Note that the following third-party packages are required on Linux:

```
python >= 2.4
gcc-c++ >= 4
fuse >= 2.6
fuse-devel >= 2.6 (RPM-based distros)
libfuse-dev >= 2.6 (DEB-based distros)
libopenssl-devel >= 0.8 (RPM-based distros)
libssl-dev >= 0.9 (DEB-based distros)
```

When done, execute

```
$> sudo make install
```

to complete the installation of XtreamFS.

## 2.5.2 Volume Management

Like many other file systems, XtreamFS supports the concept of volumes. A volume can be seen as a container for files and directories with its own policy settings, e.g. for access control and replication. Before being able to access an XtreamFS installation, at least one volume needs to be set up. This section describes how to deal with volumes in XtreamFS.

### Creating Volumes

Volumes can be created with the `xtfs_mkvol` command line utility. Please see `man xtfs_mkvol` for a full list of options and usage.

When creating a volume, it is recommended to specify the access control policy (see Sec. 2.7.2). If not specified, POSIX permissions/ACLs will be chosen by default. Unlike most other policies, access control policies cannot be changed afterwards.

In addition, it is recommended to set a default striping policy (see Sec. 2.7.4). If no per-file or per-directory default striping policy overrides the volume's default striping policy, the volume's policy is assigned to all newly created

---

files. If no volume policy is explicitly defined when creating a volume, a RAID0 policy with a stripe size of 128kB and a width of 1 will be used as the default policy.

A volume with a POSIX permission model, a stripe size of 256kB and a stripe width of 1 (i.e. all stripes will reside on the same OSD) can be created as follows:

```
$> xtrfs_mkvol -a POSIX -p RAID0 -s 256 -w 1 \  
      my-mrc-host.com:32636/myVolume
```

Creating a volume may require privileged access, which depends on whether an administrator password required by the MRC. To pass an administrator password, add `----password <password>` to the `xtrfs_mkvol` command.

For a complete list of parameters, please refer to the `xtrfs_mkvol` man pages.

## Deleting Volumes

Volumes can be deleted with the `xtrfs_rmvol` tool. Note that deleting a volume implies that *any data, i.e. all files and directories on the volume are deleted!* Please see `man xtrfs_rmvol` for a full list of options and usage.

The volume `myVolume` residing on the MRC `my-mrc-host.com:32636` can e.g. be deleted as follows:

```
$> xtrfs_rmvol my-mrc-host.com:32636/myVolume
```

Volume deletion is restricted to volume owners and privileged users. Similar to `xtrfs_mkvol`, an administrator password can be specified if required.

## Listing all Volumes

A list of all volumes can be displayed with the `xtrfs_lsvol` tool. All volumes hosted by the MRC `my-mrc-host.com:32636` can be listed as follows:

```
$> xtrfs_lsvol my-mrc-host.com:32636
```

Adding the `--l` flag will result in more details being shown.

---

### 2.5.3 Accessing Volumes

Once a volume has been created, it needs to be mounted in order to be accessed.

#### Mounting and Un-mounting

Before mounting XtreamFS volumes on a Linux machine, please ensure that the FUSE kernel module is loaded. Please check your distribution's manual to see if users must be in a special group (e.g. `trusted` in openSuSE) to be allowed to mount FUSE.

```
$> su
Password:
#> modprobe fuse
#> exit
```

Volumes are mounted with the `xtfs_mount` command:

```
$> xtfs_mount remote.dir.machine/myVolume /xtreemfs
```

`remote.dir.machine` describes the host with the Directory Service at which the volume is registered; `myVolume` is the name of the volume to be mounted. `/xtreemfs` is the directory on the local file system to which the XtreamFS volume will be mounted. For more options, please refer to `man xtfs_mount`.

Please be aware that the Directory Service URL needs to be provided when mounting a volume, while MRC URLs are used to create volumes.

When mounting a volume, the client will immediately go into background and won't display any error messages. Use the `-f` option to prevent the mount process from going into background and get all error messages printed to the console.

To check that a volume is mounted, use the `mount` command. It outputs a list of all mounts in the system. XtreamFS volumes are listed as `type fuse`:

```
/dev/fuse on /xtreemfs type fuse (rw,nosuid,nodev,user=userA)
```

Volumes are unmounted with the `xtfs_umount` tool:

```
$> xtfs_umount /xtreemfs
```

---

## Mount Options

Access to a FUSE mount is usually restricted to the user who mounted the volume. To allow the root user or any other user on the system to access the mounted volume, the FUSE options `-o allow_root` and `-o allow_other` can be used with `xtfs_mount`. They are, however, mutually exclusive. In order to use these options, the system administrator must create a FUSE configuration file `/etc/fuse.conf` and add a line `user_allow_other`.

By default, the local system cache on the client machine will be used to speed up read access to XtremFS. In particular, using the cache as a local buffer is necessary to support the `mmap` system call, which - amongst others - is required to execute applications on Linux. On the other hand, using buffered I/O may adversely affect throughput when writing large files, as FUSE  $\leq 2.7$  splits up large writes into multiple individual 4k (page size) writes. In addition, it limits the consistency model of client caches to “close-to-open”, which is similar to the model provided by NFS. Buffered I/O can be switched off by adding the `-o direct_io` parameter. The parameter effects that all read and write operations are directed to their OSDs instead of being served from local caches.

### 2.5.4 Troubleshooting

Different kinds of problems may occur when trying to create, mount or access files in a volume. In case no useful error message printed on the console, it may help to enable client-side log output. This can be done as follows:

```
$> xtfs_mount -f -d INFO /xtreemfs
```

The following list contains the most common problems and the solutions.

---

**Problem** **A volume cannot be created or mounted.**

**Solution** Please check your firewall settings on the server side. Are all ports accessible? The default ports are 32636 (MRC), 32638 (DIR), and 32640 (OSD). In case the XtreamFS installation has been set up behind a NAT, it is possible that services registered their NAT-internal network interfaces at the DIR. In this case, clients cannot properly resolve server addresses, even if port forwarding is enabled. Please check the *Address Mappings* section on the DIR status page to ensure that externally reachable network interfaces have been registered for the your servers' UUIDs. If this is not the case, it is possible to explicitly specify the network interfaces to register via the `hostname` property (see Sec. 2.4.2).

**Problem** **When trying to mount a volume, ONC-RPC exception: system error appears on the console.**

**Solution** The most common reason are incompatible protocol versions in client and server. Please make sure that client and server have the same release version numbers. They can be determined as follows:

Server: check the status pages. Alternatively, execute `rpm -qa | grep XtreamFS-server` on RPM-based distributions, or `dpkg -l | grep xtreamfs-server` on DEB-based distributions.

Client: execute `rpm -qa | grep XtreamFS-client` on RPM-based distributions, or `dpkg -l | grep xtreamfs-client` on DEB-based distributions.

**Problem** **An error occurs when trying to access a mounted volume.**

**Solution** Please make sure that you have sufficient access rights to the volume root. Superusers and volume owners can change these rights via `chmod <mode> <mountpoint>`. If you try to access a mount point to which XtreamFS was mounted by a different user, please make sure that the volume is mounted with `xtfs_mount -o allow_other ....`

---

**Problem** An I/O error occurs when trying to create new files.

**Solution** A common reason for this problem is that no OSD could be assigned to the new file. Please check if suitable OSDs are available for the volume. There are two alternative ways to do this:

- Open the MRC status page. It can be accessed via `http://<MRC-host>:30636` in the default case. For each volume, a list of suitable OSDs is shown there.
- Execute `getfattr -n xtreemfs.usable_osds ----only-values <mountpoint>`.

There may be different reasons for missing suitable OSDs:

- One or more OSDs failed to start up. Please check the log files and status pages of all OSDs to ensure that they are running.
- One or more OSDs failed to register or regularly report activity at the DIR. Please check the DIR status page to ensure that all OSDs are registered and active.
- There are no OSDs with a sufficient amount of free disk space. Please check the OSD status page to obtain information about free disk space.

**Problem** An I/O error occurs when trying to access an existing file.

**Solution** Please check whether all OSDs assigned to the file are running and reachable. This can be done as follows:

1. Get the list of all OSDs for the file: `getfattr -n xtreemfs.locations --only-values <file>`.
2. Check whether the OSDs in (one of) all replicas in the list are running and reachable, e.g. by opening the status pages or via `telnet <host> <port>`.

---

## 2.6 XtreamFS Tools

To make use of most of the advanced XtreamFS features, XtreamFS offers a variety of different tools. There are tools that support administrators with the maintenance of an XtreamFS installation, as well as tools for controlling features like replication and striping. An overview of the different tools with descriptions of how to use them are provided in the following.

### 2.6.1 Installation

When installing the XtreamFS tool suite, you can choose from two different installation sources: you can download one of the *pre-packaged releases* that we create for most Linux distributions or you can install directly from the *source tarball*.

Note that the source tarball contains the complete distribution of XtreamFS, which also includes client and server. Currently, binary distributions of the tools are only available for Linux.

#### Prerequisites

For the pre-packaged release, you will need Sun Java JRE 1.6.0 or newer to be installed on the system.

When building XtreamFS directly from the source, you need a Sun Java JDK 1.6.0 or newer, Ant 1.6.5 or newer and gmake.

#### Installing from Pre-Packaged Releases

On RPM-based distributions (RedHat, Fedora, SuSE, Mandriva, XtreamOS) you can install the package with

```
$> rpm -i xtreamfs-tools-1.1.x.rpm
```

For Debian-based distributions, please use the `.deb` package provided and install it with

```
$> dpkg -i xtreamfs-tools-1.1.x.deb
```

All XtreamFS tools will be installed to `/usr/bin`.

---

## Installing from Sources

Extract the tarball with the sources. Change to the top level directory and execute

```
$> make
```

When done, execute

```
$> sudo make install
```

to complete the installation. Note that this will also install the XtremFS client and servers.

## 2.6.2 Maintenance Tools

This section describes the tools that support administrators in maintaining an XtremFS installation.

### MRC Database Conversion

The database format in which the MRC stores its file system metadata on disk may change with future XtremFS versions, even though we attempt to keep it as stable as possible. To ensure that XtremFS server components may be updated without having to create and restore a backup of the entire installation, it is possible to convert an MRC database to a newer version by means of a version-independent XML representation.

This is done as follows:

1. Create an XML representation of the old database with the old MRC version.
2. Update the MRC to the new version.
3. Restore the database from the XML representation.

`xtfs_mrcdbtool` is a tool that is capable of doing this. It can create an XML dump of an MRC database as follows:



---

```
$> xtfs_mrcdbtool -mrc oncrpc://my-mrc-host.com:32636 \  
    dump /tmp/dump.xml
```

A file `dump.xml` containing the entire database content of the MRC running on `my-mrc-host.com:32636` is written to `/tmp/dump.xml`. For security reasons, the dump file will be created locally on the MRC host. To make sure that sufficient write permissions are granted to create the dump file, we therefore recommend to specify an absolute dump file path like `/tmp/dump.xml`.

A database dump can be restored from a dump file as follows:

```
$> xtfs_mrcdbtool -mrc oncrpc://my-mrc-host.com:32636 \  
    restore /tmp/dump.xml
```

This will restore the database stored in `/tmp/dump.xml` at `my-mrc-host.com`. Note that for safety reasons, it is only possible to restore a database from a dump if the database of the running MRC does not have any content. To restore an MRC database, it is thus necessary to delete all MRC database files before starting the MRC.

Please be aware that dumping and restoring databases may both require privileged access rights if the MRC requires an administrator password. The password can be specified via `--p`; for further details, check the `xtfs_mrcdbtool` man page.

## Scrubbing and Cleanup

In real-world environments, errors occur in the course of creating, modifying or deleting files. This can cause corruptions of file data or metadata. Such things happen e.g. if the client is suddenly terminated, or loses connection with a server component. There are several such scenarios: if a client writes to a file but does not report file sizes received from the OSD back to the MRC, inconsistencies between the file size stored in the MRC and the actual size of all objects in the OSD will occur. If a client deletes a file from the directory tree, but cannot reach the OSD, orphaned objects will remain on the OSD. If an OSD is terminated during an ongoing write operation, file content will become corrupted.

In order to detect and, if possible, resolve such inconsistencies, tools for scrubbing and OSD cleanup exist. To check the consistency of file sizes and checksums, the following command can be executed:

---

```
$> xtfs_scrub -dir oncrpc://my-dir-host.com:32638 myVolume
```

This will scrub each file in the volume `myVolume`, i.e. check file size consistency and set the correct file size on the MRC, if necessary, and check whether an invalid checksum in the OSD indicates a corrupted file content. The `-dir` argument specifies the directory service that will be used to resolve service UUIDs. Please see `man xtfs_scrub` for further details.

A second tool scans an OSD for orphaned objects, which can be used as follows:

```
$> xtfs_cleanup -dir oncrpc://localhost:32638 \  
    uuid:u2i3-28isu2-iwuv29-isjd83
```

The given UUID identifies the OSD to clean and will be resolved by the directory service defined by the `-dir` option (`localhost:32638` in this example). The process will be started and can be stopped by setting the option `-stop`. To watch the cleanup progress use option `-i` for the interactive mode. For further information see `man xtfs_cleanup`.

### 2.6.3 User Tools

Besides administrator tools, a variety of tools exist that make advanced XtremFS features accessible to users. These tools will be described in this section.

#### Showing XtremFS-specific File Info

In addition to the regular file system information provided by the `stat` Linux utility, XtremFS provides the `xtfs_stat` tool which displays XtremFS specific information for a file or directory.

```
$> cd /xtreemfs  
$> echo 'Hello World' > test.txt  
$> xtfs_stat test.txt
```

will produce output similar to the following:

---

```
filename                test.txt
XtreemFS URI            oncrpc://localhost/test/test.txt
XtreemFS fileID        41e9a04d-0b8b-467b-94ef-74ade02a2dc9:6
object type            regular file
owner                  stender
group                  users
read-only              false
```

```
XtreemFS replica list
  list version          0
  replica update policy
  -----
  replica 1 SP          STRIPING_POLICY_RAID0, 128kb, 1
  replica 1 OSDs        [{address=127.0.0.1:32640, uuid=OSD1}]
  replica 1 repl. flags 0x1
  -----
```

The fileID is the unique identifier of the file used on the OSDs to identify the file's objects. The owner/group fields are shown as reported by the MRC, you may see other names on your local system if there is no mapping (i.e. the file owner does not exist as a user on your local machine). Finally, the XtreemFS replica list shows the striping policy of the file, the number of replicas and for each replica, the OSDs used to store the objects.

## Changing Striping Policies

Currently, it is not possible to change the striping policy of an existing file, as this would require rearrangements and transfers of data between OSDs. However, it is possible to define individual striping policies for files that will be created in the future. This can be done by changing the default striping policy of the parent directory or volume.

XtreemFS provides the `xtfs_sp` tool. The tool can be used to change the striping policy that will be assigned to newly created files as follows:

```
$> xtfs_sp --set -p RAID0 -w 4 -s 256 /xtreemfs/dir
```

This will cause a RAID0 striping policy with 256kB stripe size and four OSDs to be assigned to all newly created files in `/xtreemfs/dir`.

The tool can display the default striping policy of a volume or directory as follows:

---

```
$> xtfs_sp --get /xtreemfs/dir
```

This will result in output similar to the following:

```
file:          /xtreemfs/dir
policy:        STRIPING_POLICY_RAID0
stripe-size:   4
width (kB):    256
```

When creating a new file, XtreamFS will first check whether a default striping policy has been assigned to the file's parent directory. If this is not the case, the default striping policy for the volume will be used as the striping policy for the new file. Changing a volume's or directory's default striping policy requires superuser access rights, or ownership of the volume or directory.

## Read-Only Replication

Replication is one of core features of XtreamFS. A replica can be seen as a (not essentially complete) copy of a file's content on a remote (set of) OSD(s). Replication is handled among the XtreamFS OSDs, which makes it completely transparent to client applications.

So far, XtreamFS only supports *read-only replication*. Read-only replication requires files to be immutable (i.e. 'read-only'), which implies that once a file has been replicated, it can no longer be modified. The benefit of read-only replicas is that XtreamFS can guarantee sequential replica consistency at a low cost; since files are no longer modified when replicated, no overhead is caused to ensure replica consistency.

When replicating a file, the first step is to make the file read-only, which can be done as follows:

```
$> xtfs_repl --set_readonly local-path-of-file
```

Once a file has been marked as read-only, replicas can be added. The tool supports different replica creation modes. The automatic mode retrieves a list of OSDs from the MRC and chooses the best OSD according to the current replica selection policy. You can also select a specific OSD by specifying its UUID on the command line.

Newly created replicas are initially empty, which means that no file content has been copied from other non-empty replicas. Yet, they can be immediately

---

used by applications. If a replica does not have the requested data, it fetches the data from a remote replica and saves it locally for future requests (on-demand replication). Such partial replicas help to save network bandwidth and disk usage. Alternatively, replicas can be triggered to fetch the whole data from remote replicas in the background, regardless of client requests (background replication).

Moreover, XtremFS supports different transfer strategies which has an big impact on the speed of the replication and the order in which objects are fetched. A transfer strategy must be chosen for each replica.

A replica can e.g. created as follows:

```
$> xtfs_repl --add_auto --full --strategy random \  
    /xtreemfs/file.txt
```

This command creates a new replica with an automatically-selected set of OSDs (for details, see Sec. 2.7.3, 2.6.3). The switch `--full` indicates that background replication is desired; otherwise, replicas are filled on demand, which means that they remain partial replicas until the application accesses all the objects of the replica.

To list all replicas and OSDs of the file use:

```
$> xtfs_repl -l /xtreemfs/file.txt
```

This generates output similar to this:

File is read-only.

REPLICA 1:

Striping Policy: STRIPING\_POLICY\_RAIDO

Stripe-Size: 128,00 kB

Stripe-Width: 1 (OSDs)

Replication Flags:

Complete: false

Replica Type: partial

Transfer-Strategy: random

OSDs:

[Head-OSD]          UUID: osd1, URL: /127.0.0.1:32641

REPLICA 2:

Striping Policy: STRIPING\_POLICY\_RAIDO

Stripe-Size: 128,00 kB

---

```
Stripe-Width: 1 (OSDs)
Replication Flags:
  Complete: true
  Replica Type: partial
  Transfer-Strategy: unknown
OSDs:
  [Head-OSD]      UUID: osd2, URL: /127.0.0.1:32640
```

Besides adding replicas, replicas can also be removed. Since replicas of a file do not have a fixed order, we use a replica's first OSD to identify the replica to delete. The first OSD in a replica's list of OSDs, also referred to as *head OSD* is a unique identifier for a replica, as different replicas of a file may not share any OSDs.

To remove a replica, the UUID of the head OSD must be given as an argument. It can be determined via `xtfs_repl -l`. To ensure that at least one complete replica remains, i.e. a replica that stores the entire file content, complete replicas can only be removed if there is at least one more complete replica of the file.

A replica can be removed as follows:

```
$> xtfs_repl -r osd1 /xtreemfs/file.txt
```

`osd1` refers to the UUID of the head OSD in the replica to remove.

## Automatic On-Close Replication

In addition to manually adding and removing replicas, XtremFS supports an automatic creation of new replicas when files are closed after having been initially written. This feature can e.g. be used to automatically replicate volumes that only contain write-once files, such as archival data.

To configure the behavior of the on-close replication, the `xtfs_repl` tool is used.

The number of replicas to be created when a file is closed can be specified as a volume-wide parameter, which can be set as follows:

```
$> xtfs_repl --ocr_factor_set 2 /xtreemfs
```

---

This will automatically create a second replica when the file is closed, which implies that the file will be made read-only. Note that by setting the replication factor to 1 (default value), on-close replication will be switched off, which means that the file won't be replicated and will remain writable after having been closed.

The current replication factor of a volume can be retrieved as follows:

```
$> xtfs_repl --ocr_factor_get /xtreemfs
```

Moreover, it is possible to specify whether an automatically created replica will be synchronized in the background or on demand. By default, replicas will be synced on demand. This can be changed as follows:

```
$> xtfs_repl --ocr_full_set true /xtreemfs
```

Depending on whether `--ocr_full_set` is `true` or `false`, background replication of newly created files is switched on or off.

To show whether replicas are automatically filled or not, execute the following command:

```
$> xtfs_repl --ocr_full_get /xtreemfs
```

## Changing OSD and Replica Selection Policies

When creating a new file, OSDs have to be selected on which to store the file content. Likewise, OSDs have to be selected for a newly added replica, as well as the order in which replicas are contacted when accessing a file. How these selections are done can be controlled by the user.

OSD and replica selection policies can only be set for the entire volume. Further details about the policies are described in [Sec. 2.7.3](#).

The policies are set and modified with the `xtfs_repl` tool. A policy that controls the selection of a replica is set as follows:

```
$> xtfs_repl --rsp_set dcmmap /xtreemfs
```

This will change the current replica selection policy to a policy based on a data center map. The current replica selection policy is shown as follows:

---

```
$> xtfs_repl --rsp_get /xtreemfs
```

Note that by default, there is no replica selection policy, which means that the client will attempt to access replicas in their natural order, i.e. the order in which the replicas have been created.

Similar to replica selection policies, OSD selection policies are set and retrieved:

```
$> xtfs_repl --osp_set dcmmap /xtreemfs
```

sets a data center map-based OSD selection policy, which is invoked each time a new file or replica is created. The following predefined policies exist (see Sec. 2.7.3 and `man xtfs_repl` for details):

- default
- fqdn
- dcmmap

The default OSD selection policy selects a random subset of OSDs that are responsive and have more than 2GB of free disk space, whereas the `fqdn` and `dcmmap` policies select those subsets of responsive OSDs with enough space that are closest according to fully qualified domain names and a data center map, accordingly. Besides, custom policies can be set by passing a list of basic policy IDs to be successively applied instead of a predefined policy name.

The OSD selection policy can be retrieved as follows:

```
$> xtfs_repl --osp_get /xtreemfs
```

### Setting and Listing Policy Attributes

OSD and replica selection policy behavior can be further specified by means of policy attributes. For a list of predefined attributes, see `man xtfs_repl`. Policy attributes can be set as follows:

```
$> xtfs_repl --pol_attr_set domains "*.xtreemfs.org bla.com" \  
/xtreemfs
```

A list of all policy attributes that have been set can be shown as follows:

```
$> xtfs_repl --pol_attrs_get /xtreemfs
```



---

## 2.7 Policies

Many facets of the behavior of XtremFS can be configured by means of policies. A policy defines how a certain task is performed, e.g. how the MRC selects a set of OSDs for a new file, or how it distinguishes between an authorized and an unauthorized user when files are accessed. Policies are a means to customize an XtremFS installation.

XtremFS supports a range of predefined policies for different tasks. Alternatively, administrators may define their own policies in order to adapt XtremFS to customer demands. This chapter contains information about predefined policies, as well as mechanisms to implement and plug in custom policies.

### 2.7.1 Authentication Policies

Any operation on a file system is executed on behalf of a user. The process of determining the user bound to a request is generally referred to as *user authentication*. To render user authentication customizable, the MRC allows administrators to specify an authentication policy by means of an *Authentication Provider*. Authentication Providers are modules that implement different methods for retrieving user and group IDs from requests.

The following predefined authentication providers exist:

#### UNIX uid/gid - NullAuthProvider

The NullAuthProvider is the default Authentication Provider. It simply uses the user ID and group IDs sent by the XtremFS client. This means that the client is trusted to send the correct user/group IDs.

The XtremFS Client will send the user ID and group IDs of the process which executed the file system operation, not of the user who mounted the volume!

The superuser is identified by the user ID `root` and is allowed to do everything on the MRC. This behavior is similar to NFS with `no_root_squash`.

#### Plain SSL Certificates - SimpleX509AuthProvider

XtremFS supports two kinds of X.509 certificates which can be used by the client. When mounted with a service/host certificate the XtremFS client

---

is regarded as a trusted system component. The MRC will accept any user ID and groups sent by the client and use them for authorization as with the NullAuthProvider. This setup is useful for volumes which are used by multiple users.

The second certificate type are regular user certificates. The MRC will only accept the user name and group from the certificate and ignore the user ID and groups sent by the client. Such a setup is useful if users are allowed to mount XtremFS from untrusted machines.

Both certificates are regular X.509 certificates. Service and host certificates are identified by a Common Name (CN) starting with `host/` or `xtreemfs-service/`, which can easily be used in existing security infrastructures. All other certificates are assumed to be user certificates.

If a user certificate is used, XtremFS will take the Distinguished Name (DN) as the user ID and the Organizational Unit (OU) as the group ID.

Superusers must have `xtreemfs-admin` as part of their Organizational Unit (OU).

## XtremOS Certificates - XOSAAuthProvider

In contrast to plain X.509 certificates, XtremOS embeds additional user information as extensions in XtremOS-User-Certificates. This authentication provider uses this information (global UID and global GIDs), but the behavior is similar to the SimpleX509AuthProvider.

The superuser is identified by being member of the `VOAdmin` group.

### 2.7.2 Authorization Policies

Before executing an operation, a file system needs to check whether the user bound to the operation is sufficiently authorized, i.e. is allowed to execute the operation. User authorization is managed by means of *access policies*, which reside on the MRC. Unlike authentication policies which are bound to an MRC, access policies can be defined for each volume. This has to be done when the volume is created (see `man xtfs_mkvol`). Various access policies can be used:

- **Authorize All Policy (policy Id 1)**  
No authorization - everyone can do everything. This policy is useful if performance of metadata operations matters more than security, since no evaluation of access rights is needed.

- 
- **POSIX ACLs & Permissions (policy Id 2)**  
This access policy implements the traditional POSIX permissions commonly used on Linux, as well as POSIX ACLs, an extension that provides for access control at the granularity of single users and groups. POSIX permissions should be used as the default, as it guarantees maximum compatibility with other file systems.
  - **Volume ACLs (policy Id 3)**  
Volume ACLs provide an access control model similar to POSIX ACLs & Permissions, but only allow one ACL for the whole volume. This means that there is no recursive evaluation of access rights which yields a higher performance at the price of a very coarse-grained access control.

### 2.7.3 OSD and Replica Selection Policies

When a new file is created or a replica is automatically added to a file, the MRC must decide on a set of OSDs for storing the file content. To select the most suitable subset among all known OSDs, OSD Selection Policies are used.

Replica selection is a related problem. When a client opens a file with more than one replica, the MRC uses a replica selection policy to sort the list of replicas for the client. Initially, a client will always attempt to access the first replica in the list received from the MRC. If a replica is not available, it will automatically attempt to access the next replica from the list, and restart with the first replica if all attempts have failed. Replica selection policies can be used to sort the replica lists, e.g. to ensure that clients first try to access replicas that are close to them.

Both OSD and replica selection policies share a common mechanism, in that they consist of *basic policies* that can be arbitrarily combined. Input parameters of a basic policy are a set of OSDs, the list of the current replica locations of the file, and the IP address of the client on behalf of whom the policy was called. The output parameter is a filtered and potentially sorted subset of OSDs. Since OSD lists returned by one basic policy can be used as input parameters by another one, basic policies can be chained to define more complex composite policies.

OSD and replica selection policies are assigned at volume granularity. For further details on how to set such policies, please refer to Sec. [2.6.3](#).

---

## Attributes

The behavior of basic policies can be further refined by means of policy attributes. Policy attributes are extended attributes with a name starting with `xtreemfs.policies.`, such as `xtreemfs.policies.minFreeCapacity`. Each time a policy attribute is set, all policies will be notified about the change. How an attribute change affects the policy behavior depends on the policy implementation.

## Predefined Policies

Each basic policy can be assigned to one of the three different categories called *filtering*, *grouping* and *sorting*. *Filtering policies* generate a sublist from a list of OSDs. The sublist only contains those OSDs from the original list that have a certain property. *Grouping policies* are used to select a subgroup from a given list of OSDs. They basically work in a similar manner as filtering policies, but unlike filtering policies, they always return a list of a fixed size. *Sorting policies* generate and return a reordered list from the input OSD list, without removing any OSDs.

The following predefined policies exist:

### Filtering Policies

- **Default OSD filter (policy ID 1000)**

Removes OSDs from the list that are either dead or do not have sufficient space. By default, the lower space limit for an OSD is 2GB, and the upper response time limit is 5 minutes.

Attributes:

- *free\_capacity\_bytes*: the lower space limit in bytes
- *offline\_time\_secs*: the upper response time limit in seconds

- **FQDN-based filter (policy ID 1001)**

Removes OSDs from the list that do not match any of the domains in a given set. By default, the set of domains contains '\*', which indicates that no domains are removed.

Attributes:

- 
- *domains*: a comma or space-separated list of domain names. The list may include leading and trailing '\*', which will be regarded as wildcard characters.

## Grouping Policies

- **Data center map-based grouping (policy ID 2000)**

Removes all OSDs from the OSD set that have been used in the file's replica locations list already and selects the subset of OSDs that is closest to the client and provides enough OSDs for the new replica in a single data center.

This policy uses a statically configured datacenter map that describes the distance between datacenters. It works only with IPv4 addresses at the moment. Each datacenter has a list of matching IP addresses and networks which is used to assign clients and OSDs to datacenters. Machines in the same datacenter have a distance of 0.

This policy requires a datacenter map configuration file in `/etc/xos/xtreemfs/datacentermap` on the MRC machine which is loaded at MRC startup. This config file must contain the following parameters:

- `datacenters=A,B,C`  
A comma separated list of datacenters. Datacenter names may only contain a-z, A-Z, 0-9 and `_`.
- `distance.A-B=100`  
For each pair of datacenters, the distance must be specified. As distances are symmetric, it is sufficient to specify A to B.
- `addresses.A=192.168.1.1,192.168.2.0/24`  
For each datacenter a list of matching IP addresses or networks must be specified.
- `max_cache_size=1000`  
Sets the size of the address cache that is used to lookup IP-to-datacenter matches.

A sample datacenter map could look like this:

```
datacenters=BERLIN,LONDON,NEW_YORK
distance.BERLIN-LONDON=10
distance.BERLIN-NEW_YORK=140
```

---

```
distance.LONDON-NEW_YORK=110
addresses.BERLIN=192.168.1.0/24
addresses.LONDON=192.168.2.0/24
addresses.NEW_YORK=192.168.3.0/24,192.168.100.0/25
max_cache_size=100
```

- **FQDN-based grouping (policy ID 2001)**

Removes all OSDs from the OSD set that have been used in the file's replica locations list already and selects the subset of OSDs that is closest to the client and provides enough OSDs for the new replica in a single domain.

This policy uses domain names of clients and OSDs to determine the distance between a client and an OSD, as well as if OSDs are in the same domain.

## Sorting Policies

- **Shuffling (policy ID 3000)**

Shuffles the given list of OSDs.

- **Data center map-based sorting (policy ID 3001)**

Sorts the list of OSDs in ascending order of their distance to the client, according to the data center map.

- **DNS based OSD Selection (policy ID 3002)**

The FQDN of the client and all OSDs is compared and the maximum match (from the end of the FQDN) is used to sort the OSDs. The policy sorts the list of OSDs in descending order by the number of characters that match. This policy can be used to automatically select OSDs which are close to the client, if the length of the match between two DNS entries also indicate a low latency between two machines.

## 2.7.4 Striping Policies

XtreemFS allows the content, i.e. the objects of a file to be distributed among several storage devices (OSDs). This has the benefit that the file can be read or written in parallel on multiple OSDs in order to increase throughput. To configure how files are striped, XtreemFS supports *striping policies*.

A striping policy is a rule that defines how the objects are distributed on the available OSDs. Currently, XtreemFS implements only the RAID0 policy

---

which simply stores the objects in a round robin fashion on the OSDs. The RAID0 policy has two parameters. The *striping width* defines to how many OSDs the file is distributed. If not enough OSDs are available when the file is created, the number of available OSDs will be used instead; if it is 0, an I/O error is reported to the client. The *stripe size* defines the size of each object.

Striping over several OSDs enhances the read and write throughput to a file. The maximum throughput depends on the striping width. However, using RAID0 also increases the probability of data loss. If a single OSD fails, parts of the file are no longer accessible, which generally renders the entire file useless. Replication can mitigate the problem but has all the restrictions described in Sec. 2.6.3.

## 2.7.5 Plug-in Policies

To further customize XtremFS, the set of existing policies can be extended by defining *plug-in policies*. Such policies are Java classes that implement a predefined policy interface. Currently, the following policy interfaces exist:

- `org.xtreemfs.common.auth.AuthenticationProvider`  
interface for authentication policies
- `org.xtreemfs.mrc.ac.FileAccessPolicy`  
interface for file access policies
- `org.xtreemfs.mrc.osdselection.OSDSelectionPolicy`  
interface for OSD and replica selection policies

Note that there may only be one authentication provider per MRC, while file access policies and OSD selection policies may differ for each volume. The former one is identified by means of its class name (property `authentication_provider`, see Sec. 2.4.2, 2.4.2), while volume-related policies are identified by ID numbers. It is therefore necessary to add a member field

```
public static final long POLICY_ID = 4711;
```

to all such policy implementations, where 4711 represents the individual ID number. Administrators have to ensure that such ID numbers neither clash with ID numbers of built-in policies (1-9), nor with ID numbers of other

---

plug-in policies. When creating a new volume, IDs of plug-in policies may be used just like built-in policy IDs.

Plug-in policies have to be deployed in the directory specified by the MRC configuration property `policy_dir`. The property is optional; it may be omitted if no plug-in policies are supposed to be used. An implementation of a plug-in policy can be deployed as a Java source or class file located in a directory that corresponds to the package of the class. Library dependencies may be added in the form of source, class or JAR files. JAR files have to be deployed in the top-level directory. All source files in all subdirectories are compiled at MRC start-up time and loaded on demand.



---

# Chapter 3

## The OSS API and User Guide

### 3.1 Overview

The Object Sharing Service (OSS) implements distributed objects for nodes participating in an interactive multi-user grid application. OSS runs on each client machine to enable sharing of objects residing in volatile memory. An object in this context is a replicated volatile memory region, dynamically allocated by an application or mapped into memory from a file.

Objects may contain scalars, references, and code. Therefore, OSS handles concurrent read and write access to objects and maintains the consistency of replicated objects. Persistence and security for objects stored in files are provided by XtreamFS. Fault tolerance is provided by the grid checkpointing mechanisms developed in WP3.3. OSS is being developed for Linux on IA32 or AMD64/Intel64 compatible processors.

### 3.2 Changes and New Features

This section summarizes the changes and new features between OSS releases 0.2 and 0.4.

#### 3.2.1 Object Allocators

OSS 0.4 supports different object allocators benefitting different allocation schemes and access patterns. Object allocators can be chosen on allocation of

---

objects using `oss_alloc` (see Section 3.4.1). The following object allocators exist:

- Millipage allocator: The Millipage allocator constructs special virtual-to-physical memory mappings that limit physical memory overhead in spite of false sharing avoidance.
- The MSpaces allocator is a fast and space-conserving general-purpose allocator. Being used in the GNU C library, it is very stable and tunable. It performs well if only true sharing occurs. However, for access patterns incurring false sharing, the MSpaces allocator is inferior to the Millipage allocator.
- The Linear allocator implements a first-fit allocation strategy. If objects are frequently allocated and rarely freed, the Linear allocator is a reasonable choice.
- The Page allocator only hands out objects whose size is a multiple of the page size.

See Section B.1.6 in the appendix for a description of these allocators.

### 3.2.2 Local Transaction Commits

If all objects that are accessed during a transaction are not writable on other nodes, the transaction can commit locally. It does not need to be serialized with concurrently running transactions. Therefore, the node does not need to request a commit token, such that communication with other nodes is not required. The implementation of local transaction commits distinguishes normal replicas (which improve accessibility) and backup replicas (which are used in case of node failures). At the end of a transaction, a local commit is admissible if a node has accessed only objects which do not have any normal replicas except for the local copy.

Local commits are fully transparent for the user. The OSS library interface did not need to be modified during the implementation of local commits.

### 3.2.3 Increased Flexibility

OSS release 0.4 enables the dynamic assignment of memory to nodes. In previous OSS versions, a single node could only allocate at most 32 GB of

---

memory. This limit has now been dropped. The library interface was not affected by the dynamic memory assignment.

We have also extended the monitoring subsystem, which allows precise inspection of object accesses and transaction activity. The monitoring subsystem is used only inside the library to dynamically improve allocation and replication decisions. See Section B.1.5 in the appendix for a description of the internal monitoring subsystem.

### 3.2.4 Nameservice, Overlay Network and Configuration

The OSS-internal nameservice now supports user-defined names. The nameservice runs according to transactional consistency and thus provides an easy-to-use naming scheme for internally used transactional objects.

The super-peer overlay network has been extended to support the upcoming Superpeer commit protocol.

OSS can be configured using a configuration tool similar to Linux kernel configuration. The settings that can be selected using the configuration tool are documented in Appendix B.1. Users will normally apply the default configuration. However, when wishing to test special OSS features and for library development, the tool provides a convenient way to control the configuration.

## 3.3 Installation of OSS

You can install OSS either using the prebuild distribution packages which are for example available on the XtremOS release media, or you can build and install OSS from source code. We suggest using the first method mentioned, unless you wish to configure special build-time settings for OSS.

### 3.3.1 Installing OSS using the Distribution Packages

The XtremOS release contains the OSS library, as well as a raytracing demo application to demonstrate object sharing. During the XtremOS installation procedure, simply select the checkbox *Object Sharing Service release* to install the packaged version of OSS (Library and applications). If you have XtremOS already installed and wish to install OSS, select it in the package management dialog, or run the following commands as root:

---

```
$> urpmi liboss0          (Dynamic OSS Library)
$> urpmi oss              (Raytracer and simple application)
```

Application development based on OSS need the following additional packages:

```
$> urpmi liboss0-devel    (OSS development files)
$> urpmi liboss0-static-devel (Static OSS Library)
```

### 3.3.2 Building and Installing OSS from Source

By building and installing OSS from source, you have full control over the installation process. You can configure how OSS is installed, and fine-tune all OSS features. The OSS sources can be installed from XtremOS source repository:

```
$> urpmi oss-0.4.0-1xos2.0.src (OSS sources)
```

#### Prerequisites

By building OSS from the source code, you need the following additional development packages installed on your system.

- gcc  $\geq$  4.3
- binutils  $\geq$  2.18
- make
- glibc-devel
- libglib2.0-devel  $\geq$  2.18
- libreadline5-devel

The following packages are useful to generate documentation:

- doxygen
- graphviz
- texlive

Doxygen generates source code documentation, whereas graphviz and texlive enable dependency graph and PDF file output respectively.

---

## Compilation

Unpack the OSS source code archive, change to the base directory that just has been created. The following step allows altering the default configuration of OSS (hardware architecture, features, ...) if desired. If this step is omitted, OSS will be built in its default configuration. A description of the configuration option can be found in [Appendix B.1](#).

```
$> make menuconfig
```

The following command builds the OSS library:

```
$> make
```

The make system autodetects most tools used for building OSS. If you encounter any errors, please ensure you have a recent compiler and linker installed, and that all developer packages mentioned above are installed correctly.

If you wish to pass configuration parameters via command line or to enable non-standard features, you can directly supply the corresponding parameters to make. For example, run `make -B ARCH=I686` to build OSS for 32-Bit x86 machines and `make -B ARCH=X86_64` to build OSS for 64-Bit x86 machines respectively.

## Installation

The following command installs the OSS library on the system, by default in the `/usr/local` hierarchy. For write access to system directories, you need root privileges.

```
$> make install
```

You can change the default installation hierarchy by specifying `prefix=<pathname>`, e.g. to install OSS below `/usr`, run the command

```
$> make install prefix=/usr
```

Software distributors can specify an additional prefix for the actual installation directory by defining `DESTDIR=<additional-prefix>` on the make command line.

---

### 3.3.3 Testing the OSS Installation

The OSS make system includes a command to verify that OSS has been installed correctly, and that everything needed for running a program that uses OSS is set up correctly:

```
$> make verify-install
```

The program should output the version and build information of the OSS library found according to the example below:

```
Object Sharing Service version 0.4.0 architecture I686
  subversion revision 1800 (2009-05-05 14:51:38)
build 1
Object Sharing Service has been installed correctly.
```

#### Simple test of Object Sharing

The simple test application (*oss.simple*) starts two instances of a program. The first process creates a shared object and writes the string *hello* to it. The second process waits until the object has been created, and as soon as it reads the expected string, it overwrites it with the string *world*.

#### The Raytracer Application

The raytracer is based on a application developed for a course at the MIT and has been ported to OSS with the focus on testing and demonstrating transactional shared memory. All graphical objects and the image file are allocated in transactional shared memory. Start the first node with

```
$> oss_raytracer --address <IP1>
```

and the subsequent nodes with

```
$> oss_raytracer --address <IPn> --bootstrap <IP1>
```

where *IP1* is the IP address for the first node, and *IPn* is replaced by the IP address of the respective node. To configure the tracing progress, the first node will ask some parameters:

- 
1. Consistency model: 't' for transactional consistency, 's' for strong consistency
  2. Number of Nodes
  3. Number of accesses (applies to transactional consistency only): number of accesses between transaction boundaries
  4. Pattern: specify one of 'l', 'c', 'p', 'x' or 'm'. 'l' for line by line, 'c' for column by column, 'p' for x partitions, 'x' for every Xth dot, 'm' for matching pages
  5. Scene: 1, 2, or 3
  6. Columns (e. g. 640)
  7. Rows (e. g. 480)

After rendering is done, you can give new parameters and render another scene. There are three predefined scenes in this project. *Scene1* is very simple with one sphere in the center, a few bowls around and only a few lights. *Scene2* is very complex with some arrangements of bowls and reflecting walls. *Scene3* displays the letters "OSS" consisting of bowls. You can write own scenes as *C* files analogous to *SceneDemo1.c*.

## 3.4 Developing Applications using OSS

The internal interface of the OSS library is implementation-dependent and may be extended in the future, based on insights gained during the development of OSS-based applications. In contrast, WP3.1 is currently defining an XOSAGA interface for object sharing which will eventually represent the OSS interface in a portable way.

### 3.4.1 Internal Interface of the OSS Library

The interface of the OSS library is declared in the header file `oss.h`. The following command generates an interface documentation in HTML and (if latex is available) in PDF format:

```
$> make interface-doc
```



---

The documentation is stored in the `build/doc/` subdirectory.

Let us quickly walk through the basic functionality of the OSS library. For a more detailed and precise discussion of the internal library interface, please see the Doxygen documentation generated directly from the source code. To get a deeper understanding of how to design applications that access shared objects, we suggest looking at the source code examples in the `src/apps/` subdirectory.

```
int
oss_startup(
    const char *addr,
    const char *listen_port,
    const char *bootstrap_addr,
    const char *bootstrap_port
);
```

The `oss_startup` call starts the OSS system by joining a bootstrap peer. The `addr` and `listen_port` parameters allow to bind the OSS instance to a specific interface. If no bootstrap peer is specified (i. e. a NULL pointer is passed), a new distributed object storage is created. A return value of zero indicates successful startup.

```
void *
oss_alloc(
    size_t size,
    oss_consistency_model_t consistency_model,
    oss_alloc_attributes_t *attributes
);
```

The `oss_alloc` call creates a shared object of specified size, initializes it with a consistency model and further attributes, and returns an identifier for the object. In the attributes structure, one can select a specific allocator. Currently, the following allocators are available: `oss_page_allocator`, `oss_mspaces_allocator`, `oss_millipage_allocator` and `oss_list_allocator`.

```
void
oss_free(
    void *ptr
);
```

The `oss_free` call frees some memory which has previously been dynamically allocated using `oss_alloc`.

---

```
oss_transaction_id_t
oss_bot(
    oss_transaction_priority_t priority,
    oss_transaction_attributes_t *attributes
);
```

The `oss_bot` call marks the begin of a transaction with given priority and attributes. OSS guarantees that all accesses to distributed objects between `oss_bot` and `oss_eot` perform atomically, consistent, isolated, and durable. The return value references the transaction that has been started, or equals `oss_undefined_transaction_id` which indicates that the transaction failed to start.

```
int
oss_eot(
    oss_transaction_id_t taid
);
```

The `oss_eot` call denotes the end of the supplied transaction.

```
int
oss_abort(
    oss_transaction_id_t taid
);
oss_permit_abort(
    oss_transaction_id_t taid
);
```

Both calls handle voluntarily aborting a transaction. An application that somehow finds out that it cannot commit, or that committing will have adverse effect, may call `oss_abort` to unconditionally abort the supplied transaction. Depending on the transaction attributes used, the transaction will restart or simply fail. An application may optionally call `oss_permit_abort` to mark locations in the code where it is safe to abort a transaction. If the transaction is already known to fail on commit, OSS can restart the transaction and need not delay restarting the transaction until `oss_eot`. If the success of the transaction is not yet determined, the call to `oss_permit_abort` will simply appear as a void statement.

---

```
void *
oss_nameservice_get(
    const char *id
);
```

```
void
oss_nameservice_set(
    const char *id,
    void *val
);
```

OSS contains a simple name service, which applications can use to store and retrieve object IDs. The name service has a tree structure, with slashes (/) separating directory levels. Each entry begins with a slash. An application or OSS module can set a value for a name by calling `oss_nameservice_set` and retrieve a value by calling `oss_nameservice_get`. A value that has not yet been set is treated as object ID NULL.

```
void
oss_wait(
    void *addr,
    unsigned char value
);
```

The barriers implementation allows applications to wait until the character object pointed to by `addr` contains a target value. The character object may be subject to transactional or strong consistency.

### 3.4.2 Linking against the OSS Library

The OSS library is built as a static shared library (`liboss.a`) and as a dynamic shared library (`liboss.so`). Simply specify the option `-loss` to the compiler driver or linker, which will link against the appropriate static or dynamic library. If you did not install the library into a well-known location such as `/usr/lib`, you will need to specify the path to the library via the option `-L<path>`.

# Appendix A

## XtreemFS Appendix

### A.1 XtreemFS Support

Please visit the [XtreemFS website at www.xtreemfs.org](http://www.xtreemfs.org) for links to the user mailing list, bug tracker and further information.

### A.2 XtreemOS Integration

#### A.2.1 XtreemFS Security Preparations

XtreemFS can be integrated in an existing XtreemOS VO security infrastructure. XtreemOS uses X.509 certificates to authenticate users in a Grid system, so the general setup is similar to a normal SSL-based configuration.

Thus, in an XtreemOS environment, certificates have to be created for the services as a first step. This is done by issuing a *Certificate Signing Request (CSR)* to the RCA server by means of the `create-server-csr` command. For further details, see the Section Using the RCA in the XtreemOS User Guide.

Signed certificates and keys generated by are RCA infrastructure are stored locally in PEM format. Since XtreemFS services are currently not capable of processing PEM certificates, keys and certificates have to be converted to PKCS12 and Java Keystore format, respectively.

Each XtreemFS service needs a certificate and a private key in order to be run. Once they have created and signed, the conversion has to take place. Assuming that certificate/private key pairs reside in the current working

---

directory for the Directory Service, an MRC and an OSD (`ds.pem`, `ds.key`, `mrc.pem`, `mrc.key`, `osd.pem` and `osd.key`), the conversion can be initiated with the following commands:

```
$> openssl pkcs12 -export -in ds.pem -inkey ds.key \  
    -out ds.p12 -name "DS"  
$> openssl pkcs12 -export -in mrc.pem -inkey mrc.key \  
    -out mrc.p12 -name "MRC"  
$> openssl pkcs12 -export -in osd.pem -inkey osd.key \  
    -out osd.p12 -name "OSD"
```

This will create three PKCS12 files (`ds.p12`, `mrc.p12` and `osd.p12`), each containing the private key and certificate for the respective service.

XtreemFS services need a *trust store* that contains all trusted Certification Authority certificates. Since all certificates created via the RCA have been signed by the XtreamOS CA, the XtreamOS CA certificate has to be included in the trust store. To create a new trust store containing the XtreamOS CA certificate, execute the following command:

```
$> keytool -import -alias xosrootca -keystore xosrootca.jks \  
    -trustcacerts -file \  
    /etc/xos/truststore/xtreemosrootcacert.pem
```

This will create a new Java Keystore `xosrootca.jks` with the XtreamOS CA certificate in the current working directory. The password chosen when asked will later have to be added as a property in the service configuration files.

Once all keys and certificates have been converted, the resulting files should be moved to `/etc/xos/xtreemfs/truststore/certs` as root:

```
# mv ds.p12 /etc/xos/xtreemfs/truststore/certs  
# mv mrc.p12 /etc/xos/xtreemfs/truststore/certs  
# mv osd.p12 /etc/xos/xtreemfs/truststore/certs  
# mv xosrootca.jks /etc/xos/xtreemfs/truststore/certs
```

For setting up a *secured* XtreamFS infrastructure, each service provides the following properties:

---

```
# specify whether SSL is required
ssl.enabled = true

# server credentials for SSL handshakes
ssl.service_creds = /etc/xos/xtreemfs/truststore/certs/\
service.p12
ssl.service_creds.pw = xtreemfs
ssl.service_creds.container = pkcs12

# trusted certificates for SSL handshakes
ssl.trusted_certs = /etc/xos/xtreemfs/truststore/certs/\
xosrootca.jks
ssl.trusted_certs.pw = xtreemfs
ssl.trusted_certs.container = jks
```

`service.p12` refers to the converted file containing the credentials of the respective service. Make sure that all paths and passphrases (`xtreemfs` in this example) are correct.

## A.3 XtreamFS Command Line Utilities

**xtfs\_cleanup** Deletes orphaned objects on an OSD and restores orphaned files.

**xtfs\_lsvol** Lists the volumes on an MRC.

**xtfs\_mkvol** Creates a new volume on an MRC.

**xtfs\_mount** The XtreamFS client which mounts an XtreamFS volume locally on a machine.

**xtfs\_mrcdbtool** Dumps and restores an XML representation of the MRC database.

**xtfs\_repl** Controls file replication in XtreamFS.

**xtfs\_rmvol** Deletes a volume.

**xtfs\_sp** Displays and modifies default striping policies for directories and volumes.

---

**xtfs\_scrub** Examines all files in a volume for wrong file sizes and checksums and corrects wrong file sizes in the MRC.

**xtfs\_stat** Displays XtremFS-specific file information, such as OSD lists and striping policies.

**xtfs\_test** Automatically sets up an XtremFS testing environment and runs the automatic XtremFS test suite.

**xtfs\_umount** Un-mounts a mounted XtremFS volume.

# Appendix B

## OSS Appendix

### B.1 OSS Configuration Options

This chapter describes all configuration options of OSS, which affect compilation of OSS. The configuration dialog is accessible via

```
$> make menuconfig
```

#### B.1.1 Debugging

Library developers can configure a number of debugging options.

##### **debug level for whole build process**

Selects a global level for debug output unless this value is overridden by a *per-file debug level*.

##### **debug glib**

Enables debug output for glib related operations.

##### **debug networking**

Enables debug output for network related operations.



---

## Per-file debug levels

Allows a fine granular debug level selection for specific source files.

## B.1.2 Code generation

The binary code of the OSS library can be compiled for different processor architectures.

### Processor Architecture

Defines the processor architecture for which OSS is compiled. OSS supports the following architectures:

- AMD64/Intel64 architecture (64-bit operating system provided)
- I686 architecture (32-bit or 64-bit<sup>1</sup>supported)

## B.1.3 Library Interface

In addition to the base functions which the OSS library always exports, a number of functions are tagged as optional or experimental.

### **oss\_mmap**

Exports the command `oss_mmap` which creates an object from the content of a file, and the commands `oss_munmap` and `oss_msync`, which will unmap and synchronize object and file in a future version of OSS.

### **oss\_sync/oss\_push/oss\_pull**

*Not implemented yet.*

### **oss\_nameservice\_get/oss\_nameservice\_set**

Exports the functions of the nameservice to the API. This allows applications to use the nameservice of OSS.

---

<sup>1</sup>The usage of a 32-bit OSS version in an 64-bit XtreamOS system needs the installation of a 32-bit compatibility layer (32-bit libraries).

---

### **nameservice consistency**

Selects the consistency model of nameservice entries. Some internally defined entries are always handled according to strong consistency.

### **miscellaneous debug functions**

Exports further debugging functions to the API (*see oss.h*).

### **unstable library interface**

Exports functions for retrieving the own node id, the number of nodes, and setting the number of nodes participating on transactional consistency to the API (*see oss.h*). These functions are used for debugging purposes only. (*Without claim to be still available in future versions of OSS*).

### **oss\_wait**

Enables distributed barriers for strong and transactional consistency.

## **B.1.4 Communication**

The OSS library interface deliberately does not specify how nodes are interconnected. Internal to the OSS library, node interconnection can be implemented in several ways.

### **Overlay Routing**

Allows configuration of overlay network related options. Unless selected, the node network is fully meshed; however, connections are established on demand.

### **Superpeer Network**

Enables routing of OSS messages in the overlay network [**Experimental**].

---

## B.1.5 Monitoring

For performance measurements as well as for automatic reconfiguration during runtime, the library contains a monitoring subsystem. The subsystem allows the library developer to intersperse monitoring events in the source code. Different handlers can be attached to monitoring events by specifying their names in the configuration dialog. The default no-op handler is called `null`. The `count` handler simply counts the number of events. The `printf` handler prints the events seen immediately, including the source code location and a custom pointer value. The `latency` handler measures the duration of events, whereas the `slist` handler accumulates the pointer values of all events seen in a singly-linked list.

### **monitoring**

Enables monitoring of several OSS internal operations for statistics and dynamic reconfiguration.

### **log monitor data to file**

Enables logging the monitoring data to a file. Unless selected, the monitors print statistics to standard output.

### **periodic dump**

Time interval in seconds of periodic monitoring data dump.

### **short log**

Reduces verbosity of logging information output.

### **object\_mmap**

Monitors object mappings.

### **object\_alloc**

Monitors object allocations.

---

**object\_free**

Monitors object deallocations.

**read\_fault**

Monitors detected read accesses.

**write\_fault**

Monitors detected write accesses.

**read\_access**

Monitors read accesses evoked by a test application.

**write\_access**

Monitors write accesses evoked by a test application.

## **B.1.6 Memory allocator**

OSS supports different memory allocators.

**mSPACE allocation from dmalloc**

Enables the mspaces memory allocator. The mspace allocator is a general-purpose allocator, which is very reliable and versatile.

**millipage implementation**

Enables the millipage memory allocator. This allocator concentrates multiple objects allocated on different memory pages on one physical page frame. The millipage allocator is well suited for allocations of small objects if another allocator might induce false sharing.

---

### **simple list allocator**

Enables the simple first-fit memory allocator. The simple list allocator is very fast for allocations, but frequent deallocations may induce external fragmentation.

### **replica management**

Currently, replication is handled using invalidations and requests for invalid objects. A future release of the library will include a full-featured replica management that handles a combination of object invalidations and updates.

### **diff computation and transfer**

Diff computation and transfer will speed up object accesses, but it is still under development and not included in the current release.

## **B.1.7 Applications**

### **build raytracer**

Builds the raytracer application, shipped with OSS.

### **build wissenheim**

Builds the wissenheim application out of OSS. This option is only intended for debugging purposes regarding Wissenheim over OSS.

## **B.1.8 Remote installation**

UDUS infrastructure specific options (*not for public usage*).

# Index

- Access Policy, 43
  - Authorize All, 43
  - POSIX ACLs, 44
  - POSIX Permissions, 44
  - Volume ACLs, 44
- allow\_others option, 29
- allow\_root option, 29
- Architecture, 5
- Authentication, 4
- Authentication Provider, 9, 42
  - NullAuthProvider, 42
  - SimpleX509AuthProvider, 42
  - XOSAuthProvider, 43
- Authorization, 4
- Authorize All Access Policy, 43
- CA
  - Certificate Authority, 11
- Certificate, 5, 10
- Certificate Authority, 11
- Client, 6
- Create Volume, 26
- Credentials, 10
- Delete Volume, 27
- DIR, 5
- Directory Service, 5
- fileID, 36
- FUSE, 6
- init.d, 22
- Java KeyStore, 11
- JKS, 11
- Metadata, 5
- Metadata and Replica Catalog, 5
- Metadata Server, 5
- Mount, 28
- Mounting, 6
- MRC, 5
- NullAuthProvider, 42
- Object, 5
- object, 51
- object sharing service, 51
- Object Storage Device, 5
- Object-based File System, 5
- On-close Replication, 39
- OSD, 5
- OSD Selection Policy, 44
- OSS, 51
- PKCS#12, 10
- Policy
  - Access Policy, 43
  - OSD Selection Policy, 44
  - Striping Policy, 5, 47
- POSIX ACLs Access Policy, 44
- POSIX Permissions Access Policy, 44
- RAID0, 4, 47
- raytracer application, 56
- Read-only Replication, 37
- Replication, 37, 39
  - on-close, 39
  - read-only, 37
- SAGA, 57

---

SimpleX509AuthProvider, [42](#)  
SSL, [5](#)  
Status Page, [23](#)  
Storage Server, [5](#)  
Stripe Size, [48](#)  
Striping, [47](#)  
    Stripe Size, [48](#)  
Striping Policy, [5](#), [47](#)  
Striping Width, [48](#)  
  
Unmount, [28](#)  
user\_allow\_other option, [29](#)  
UUID, [8](#)  
  
VFS, [6](#)  
Volume, [5](#), [6](#)  
    Create, [26](#)  
    Delete, [27](#)  
    Mount, [28](#)  
    Un-mount, [28](#)  
Volume ACLs Access Policy, [44](#)  
  
X.509, [5](#), [10](#)  
XOSAGA, [57](#)  
XOSAuthProvider, [43](#)  
xtfs\_mkvol, [26](#)  
xtfs\_mount, [28](#)  
xtfs\_rmvol, [27](#)  
xtfs\_sp, [36](#)  
xtfs\_stat, [35](#)  
xtfs\_umount, [28](#)  
XtreemFS stat, [35](#)  
XtreemFS striping policy tool, [36](#)  
XtreemOS  
    Integration, [61](#)  
    XtreemOS Certificates, [43](#)