The OFA Standard—Oracle for Open Systems

Cary V. Millsap
Oracle Corporation

September 24, 1995

The OFA Standard is a set of configuration guidelines that will give you faster, more reliable Oracle databases that require less work to maintain. The OFA Standard is written by the founder of the Oracle team responsible for installing, tuning, and upgrading several hundreds of sites worldwide since 1990—this paper is based on the best practices of those hundreds of sites. Today the "Optimal Flexible Architecture'' described in the OFA Standard is built into the Oracle configuration tools and documentation on all open systems ports.

This paper formally defines the OFA Standard for configuring complex Oracle systems at sites demanding high performance with low maintenance under continually evolving requirements. It also details how the OFA is derived from requirements essential to successful implementation of complicated software on any system. Along with the definition of the OFA, this paper will also reveal the strategy and analysis that motivate the individual recommendations of Oracle Services' Optimal Flexible Architecture. By reading this paper, you will more fully understand the challenges that confront the Oracle Server configuration planner.

Contents

- Attempting to organize large amounts of complicated software and data on disk leads to device bottlenecks and poor performance.
- Routine administrative tasks like software and data backup are performed incorrectly or inefficiently, causing vulnerability to corruptive influences.
- Attempting to switch among multiple Oracle databases results in corruption of production data.
- Inability to adequately administer database segment growth results in recurrent application failures.

Figure 1. These configuration problems motivated the original OFA.

Introduction

At the 1991 International Oracle User Week in Miami, Oracle Services presented a paper describing a low maintenance way to configure high performance, growing Oracle databases [Millsap 1993]. The Optimal Flexible Architecture (OFA) described at that presentation had already helped solve devastating problems at several production Oracle sites. outlines some of the problems that motivated the original OFA work. The promise of the OFA was that an entire class of problems could be avoided by applying tested knowledge of how to optimize Oracle's relationship with its host operating system (Figure 1).

A large worldwide audience welcomed the OFA because it delivered on its promise. Existing installations adopted it because their configuration problems were taking money from their bottom lines. New customers welcomed the OFA because it filled a desperate need for knowledge in areas that only a handful of specialists in the world knew anything about. Oracle Services' Optimal Flexible Architecture was the first widely published document that made specific configuration recommendations based on actual field experience. The OFA has since become the world's most popular specification for configuring high performance, low maintenance Oracle database systems.

The original OFA paper described how to exploit the capacity of modern operating systems to organize massive amounts of data and, generally, how to make configuration decisions that minimize the cost of administering a database. In 1992, Oracle Services published a comprehensive technical OFA Standard for UNIX that addressed a broad range of configuration issues including: naming standards, file protections, UNIX logins, raw devices, revision management, storage parameters, resource privileges, and rollback segments. This document was distributed only to a few configuration specialists, but the work motivated several Oracle product and documentation refinements. By early 1993 Oracle had integrated the OFA Standard into Oracle7.

A good standard should act as a strong floor, without becoming a ceiling that inhibits "creative magic." Creating a useful standard requires its author to master the precarious balance between appropriate flexibility and actual usefulness to people who just want to know what to type. In an attempt to achieve this balance, *The OFA Standard—Oracle for Open Systems* identifies universal requirements to motivate a generic standard, which is then illustrated with a specific example. The resulting document is intended to be useful both as a configuration

```
#device            device              mount  FS    fsck  mount   mount
#to mount          to fsck             point  type  pass  atboot  options
/dev/dsk/c0t3d0s0  /dev/rdsk/c0t3d0s0  /      ufs   1     yes     -
/dev/dsk/c0t3d0s6  /dev/rdsk/c0t3d0s6  /usr   ufs   2     yes     -
```

Figure 2. This Solaris **/etc/vfstab** excerpt shows the mapping of mount point names to device names. Each row in this file represents one disk *slice*.

requirements definition for sites of all sizes and as the formal definition of the OFA Standard.

This paper does not address all the questions you will encounter as you plan your configuration. However, we hope that by studying the OFA Standard, you will understand the importance of investing care into the optimal and flexible configuration of the system to which you will entrust your strategic data. We hope also that you will want to keep learning after having read this monograph, and that you will give our team the opportunity to execute our mission: To help you make the most efficient use of the resources you pay for by sharing knowledge with you at your own site.

1. Operating System Configuration

If you are interested in installing a relational database management system, it is likely that you will be installing the largest application your computer is capable of supporting. Before your system can serve your Oracle application, you must connect and configure peripherals, configure disks for swap space and data storage, name your file systems, configure your network, prepare your UNIX kernel for the specific demands your new applications will have, and create logins [Frisch 1991, Loukides 1991, Nemeth et al 1989]. The following sections will assist you and your systems engineer to understand some of the issues your Oracle configuration planner must consider.

1.1 Mount Points

One of the first tasks undertaken to permanently configure a UNIX system is to select sizes and names for file system mount points. A *mount point* is a directory name denoting where the file subsystem for a single disk slice will be linked into an existing UNIX file system. The sample **/etc/vfstab** excerpt from a Solaris system in Figure 2 shows the mapping of two mount points, **/** and **/usr**, to two disk slices. Although physically the contents of these two directories live in parallel slices on one disk, a UNIX user regards **usr** hierarchically as a subdirectory of **/**.

Regardless of whether you use “raw devices’’ for Oracle data (sec. 4), you will have to choose mount point names for slices containing users’ files and software. The file system makes UNIX easier to administer by hiding device details, and selection of mount point names sets the stage for exactly how easy it will be. In selecting names, the configuration planner must find an appropriate balance among these important requirements:

> *Requirement 1.* The file system must be organized so that it is easy to administer growth from: adding data into existing databases, adding users, creating databases, and adding hardware.
>
> *Requirement 2.* It must be possible to distribute I/O load across sufficiently many disk drives to prevent a performance bottleneck.
>
> *Requirement 3.* It may be necessary to minimize hardware cost.
>
> *Requirement 4.* It may be necessary to isolate the impact of drive failure across as few applications as possible.

The way to balance requirements 2 and 3 is to name every one of your UNIX mount points in such a manner that it is possible to lay a file from any database there without violating requirement 1. The following rule accomplishes this balance.

OFA 1 *Name all mount points that will hold site-specific data to match the pattern* ***/pm*** *where* ***p*** *is a string constant chosen not to misrepresent the contents of any mount point, and* ***m*** *is a unique fixed-length key that distinguishes one mount point from another.*

If files from two or more applications will live together in a mount point subtree, it is important that the mount point name not denote the presence of one application to the exclusion of another. For example, consider a situation in which you would like to balance I/O load by storing both an Oracle database file and an online backup of some other data within a single disk slice. You shouldn't call that disk slice's mount point **/oracle** because that would put a misleading name in the path of the non-Oracle files stored there. And you shouldn't call the mount point **/backup** because you wouldn't want to keep Oracle database files in a directory that looks like it has been made exclusively for storage of backed up files.

Mount points that contain files from multiple applications are given names only to distinguish one mount point from another, like **/u01** and **/u02**, or even **/ulna/disk01** and **/ulna/disk02**. Using zeros to pad distinguishing keys to a fixed length makes sorted lists of mount point names come out right, like

```
  ...                    /u1
/u08                     /u10
/u09      instead of     /u11
/u10                     /u2
  ...                    ...
```

Unfortunately, storing files from two or more databases on the same drive contradicts requirement 4. The only way to satisfy requirements 2 and 4 simultaneously is to buy more disk drives than your initial sizing estimates probably showed. Administrators with tight hardware budget constraints normally sacrifice maximal fault resilience (req. 4), because performance (req. 2) is almost always more important. So, the degree to which you are willing to solve this conflict with hardware quantity determines the strategy you should take for naming your mount points. An alternative strategy for the small percentage of Oracle sites that can pursue fault resilience without sacrificing performance is given in section 6 in this paper.

Resist the temptation to encode controller or disk identification characters into a mount point name. Putting specific hardware configuration detail into directory names causes exactly the types of problems that mount point names were designed to avoid. Mount point names allow us to "abstract away'' the details of hardware implementation that are irrelevant to the challenges that a system's users face.

Administrators can be tempted to match mount point names with device names because, to balance I/O load, they have to make decisions about directory contents based on I/O statistics printed for devices. However, denoting hardware information within a mount point name causes trouble if you ever upgrade your disk hardware. New hardware that uses different device names than you had before (consider the case of exchanging a SCSI drive for an IPI drive) will require you to invest time into changing path names in your applications if you intend to follow your own hardware-bound naming tradition. Your best mount point naming decision is to use names unrelated to your hardware device names and to examine system files (or write a program to do it for you) on the rare occasion when you need to balance your system's I/O load.[1]

1.2 Login Home Directories

On very old UNIX systems, home directories were placed in **/usr**. That worked well enough for single-user systems, but having home directories in **/usr** did cause unnecessary challenges. For example, having directories in **/usr** increases risk of accidental file loss at upgrade time when the entire **/usr** subtree is replaced; also, if the **/usr** file system fills, UNIX will crash. UNIX books today usually recommend that login home directories be placed in a directory called **/u** or **/home**. Using **/u** or **/home** works fine for sites with average data volume. However, a challenge arises if the set of files to be contained within a single home directory is too large to fit on any single disk slice. For example, Oracle Financials and Manufacturing software consumes almost a gigabyte, all of which is supposed to reside in some directory called **applmgr**, but no single disk slice on your sys-

[1] The requirement implicit here is very similar to requirement 6 (to which you will be introduced shortly): It must be possible to exchange hardware components without having to revise programs that refer to them.

```
$ ln -s /u03/app/applmgr/gl /u02/app/applmgr/gl
$ ls -l /u02/app/applmgr
-rw-r--r--  1 applmgr       1119 Jul  5 01:16 AOL.env
drwxrwxr-x  2 applmgr       2048 Jul  5 01:16 alr
drwxrwxr-x  2 applmgr       2048 Jul  5 01:16 fnd
lrwxrwxrwx  1 applmgr          5 Jul  5 01:16 gl -> /u03/app/applmgr/gl
...
$ ls -l /u03/app/applmgr
drwxrwxr-x  1 applmgr       2048 Jul  5 01:16 gl
```

Figure 3. In this figure, we use a symbolic link to distribute the **applmgr** login's files across disks. The **gl** directory in **/u02/app/applmgr** is actually a named (symbolic) link referring to a directory subtree that is physically located on the **/u03** mount point. Thus we can treat the **gl** directory as if it were a part of **/u02** in our administrative programming.

tem is big enough to hold all of it. Hence, the following requirement:

> *Requirement 5.* It must be possible to distribute across two or more disk drives both (a) the collection of home directories and (b) the contents of an individual home directory.

The following rule offers an elegant solution to requirement 5:

OFA 2 *Name home directories matching the pattern* ***/pm/h/u****, where* ***pm*** *is a mount point name,* ***h*** *is selected from a small set of standard directory names, and* ***u*** *is the name of the owner of the directory.*

Placing all home directories at the same level in the UNIX file system means that we can put home directories on different mount points, yet still be able to refer to the collection of login homes with a single pattern. We meet requirement 5.a without violating requirement 1 by placing two large home subtrees at the same level on separate mount points. For example, the Oracle Server software owner home directory might be **/u01/app/oracle**, and the Oracle Applications software owner home directory might be **/u02/app/applmgr**. Even though two enormous subtrees are on different disk slices, we can still use a single pattern—in this example, **/*/app/***—to refer to all applications owner login home directories on the system.

To illustrate requirement 5.b, consider the Oracle Financial and Manufacturing Applications software owner, typically named **applmgr**, which can own more than a gigabyte of UNIX files on a system whose largest disk slice is 600 megabytes. A simple solution is to use symbolic links to make directories appear in a single subtree, even though they physically reside on different mount points. Figure 3 shows the symbolic link required to enable the Oracle General Ledger software to live on a separate mount point from the other applications software, yet appear to live in **/u02/app/applmgr**. All **applmgr** files are still identifiable as residents of subtrees whose names match the pattern **/*/app/applmgr**.

Different values of ***h*** in rule OFA 2 enable system administrators to simplify their backup procedures by using different home directory roots for different types of users. For example, Oracle Server software files might be owned by a UNIX login called **oracle**, just like the file **résumé.tex** might be owned by a login called **cmillsap**. However, the administrator of the UNIX system housing both users may wish to back up the two types of files represented here on different tapes or different schedules. To partition a system's users into the two classes (1) applications owners and (2) interactive logins, the administrator could choose home directory subtree names with, for example, ***h*** chosen from the list {**app**, **home**}. For example, you might use the following UNIX command to back up your applications software directories:

```
find /*/app/* -print | \
bar cvf /dev/rst0
```

…And this UNIX command to back up your login user home directories:

```
find /*/home/* -print | \
bar cvf /dev/rst0
```

1.3 User Profiles

Oracle Server is designed well to enable users to choose which of several simultaneously active versions of server software to run against any of several databases, without sophisticated programming and without complicated user profiles. An Oracle database user's profile should (1) assign a UNIX path value so that the user's shell can execute Oracle's programs for setting up the environment; (2) define an instance name (SID) for planned database connections; and (3) execute the program that sets the UNIX path for running Oracle software. Insert the following lines into your users' **.profile** to accomplish these tasks:[2]

```
# set UNIX path
PATH=/bin:/usr/bin:/var/opt/bin

# set default instance
ORACLE_SID=sab

# set ORACLE_SID, ORACLE_HOME,
# and PATH without asking
ORAENV_ASK=NO
. oraenv
ORAENV_ASK=
```

By ensuring that **oraenv**, **coraenv**, and **dbhome** reside in **/var/opt/bin**,[3] independent of any Oracle software home directory, you remove all dependence on Oracle versions from your login profiles.

The steps shown here should be executed for each user connecting to an Oracle Server instance. A user may want to perform additional setup steps in the profile, such as assigning terminal settings, inserting additional software directories in the UNIX path, assigning **ORACLE_PATH** and additional environment variables, or selecting among various instances for connection.

1.4 Zero Maintenance Administration

Oracle Services still occasionally visits an unfortunate client whose backup programs didn't keep pace with the file system changes made when someone cured an I/O bottleneck or ferreted out free space to feed a growing application. We have been asked to visit several sites at which a routine I/O balancing exercise that should have consumed ten minutes actually consumed hours of database downtime because administrative programs contained hard-coded references to path names that were no longer valid after the surgery. These kinds of problems motivate the following requirement.

> *Requirement 6.* It must be possible to add or move login home directories without having to revise programs that refer to them.

Conforming to the following rule satisfies requirement 6.

OFA 3 *Refer to explicit path names only in files designed specifically to store them, such as the UNIX* ***/etc/passwd*** *file and the Oracle* ***oratab*** *file; refer to group memberships only in* ***/etc/group****.*

Hard-coded references to a file's path name must be systematically identified and modified if it ever becomes necessary to relocate that file to a new directory. Fortunately, it is completely unnecessary to record a path name in any file other than a central reference file, because a user's home directory name can be derived. C Shell and KornShell users can use ***~login*** to refer to the home directory for ***login***; and although the Bourne shell has no built-in ability to calculate home directory names, it is easy to construct a program to do it. Such a program is given later in this document (**lhd**, section 9).

Similarly, group memberships should never be recorded in an administrative program, because group member names can be computed from **/etc/group**, as is also shown later (**grpx**, section 9). By always using programs like **lhd** (or **~**) and **grpx** instead of explicit references, your administrative tools will not require modification when you move user home directories or change group memberships.

---

[2] The example shown here is what you will need for Bourne shell or KornShell users. For C Shell users, you will need to make functionally equivalent entries into the **.cshrc** file for each user.

[3] The standard directory for site local programs is called **/usr/local/bin** or **/usr/lbin** on most systems before UNIX System V Release 4.

```
# Back up all files in login home subtrees of the 'applmgr' user
find /*/app/applmgr -print | tar cvf /dev/rst0

# Propagate a centrally administered .profile to 'clerk' users
for user in `grpx clerk` ; do
        f=`lhd $user`/.profile
        ln -s `lhd applmgr`/.profile.clerk $f
        chown applmgr $f ; chgrp oaa $f ; chmod 644 $f
done
```

Figure 4. This figure shows an example of zero maintenance programming. Regardless of the physical location of the **applmgr** subtrees on the system, the first **find** command will find them because of the wildcard in the mount point directory name component of the path. The second UNIX command will create a properly protected file into each member of the **clerk** group. Neither of these programs require modification if login home directories are added, moved, or deleted. (Source code for the **grpx** and **lhd** programs used here is supplied later in this paper.)

Figure 4 illustrates programming that exploits the ability to refer to classes of objects with simple patterns that remain constant when you add data, users, databases, or disks; even when you move files around your system. Consider the change you would have to make to the one-line backup program shown here if you were to change the physical location of the **applmgr** user's login home directory. How would adding a new user to the **dba** or **clerk** group affect the behavior of the second or third example? The OFA Standard ensures that no maintenance on these programs would be required to accommodate these changes. You will see later (Figure 7) that using the OFA also gives a zero maintenance way to manipulate all the files associated with a given Oracle database as a unit, even though the files might be distributed uniformly across many disks.

2. Oracle Files

Before the original OFA recommendations were published, many people placed database files in **$ORACLE_HOME/dbs** [Millsap 1993]. However, putting control files, redo log files, or data files in a subtree of the directory holding Oracle software caused problems: having all database files on a single disk bottlenecked the server; and having long-term database data in the Oracle software subtree made upgrades unnecessarily difficult because administrators had to spend time and money to carefully preserve and move the data. A goal of the original OFA was to solve these difficult problems with a flexible set of recommendations that allowed files to be distributed across multiple disk drives without disorganization, yet prevented software and data from interfering with each other during software upgrades. The separation of data from software is just one specific case of a broader requirement:

> *Requirement 7.* Categories of files must be separated into independent directory subtrees so that files in one category are minimally affected by operations upon files in the other categories.

The following classification of Oracle files enables us to build a standard that meets this requirement:

- *Product files* consist of Oracle Server software and tools that are supplied on the Oracle Corporation distribution media.
- *Administrative files* are files containing data about the database or instance, including archived redo log files, server process diagnostic output, database creation scripts, online exports, instance parameter files, etc.
- *Local software* is software used with Oracle that is written on site or purchased separately from the Oracle distribution software.
- *Database files* consist of control files, redo log files, and data files.

This classification partitions Oracle files along the boundaries of different lifespans, maintenance schedules, and security privileges. The following sections describe a model that fulfills requirement 7 by placing each of these four

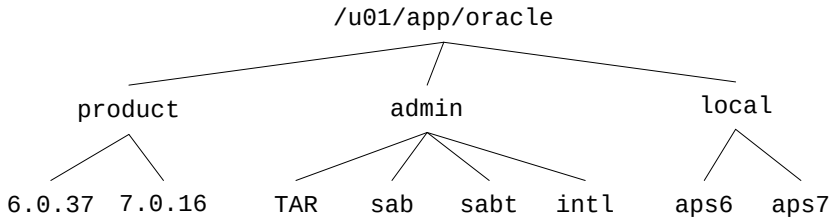

Figure 5. This is a graphical representation of the Oracle software owner's home directory structure. In this example, **ORACLE_BASE** would be set to the value **/u01/app/oracle**, and **ORACLE_HOME** would be set to either **/u01/app/oracle/product/6.0.37** or **/u01/app/oracle/product/7.0.16**.

types of files in subtrees independent of the other types.

2.1 Oracle Software and Administrative Data

Figure 5 is a graphical representation of the OFA Standards that will be detailed below for storing Oracle product, administrative, and local software files. All Oracle software and administrative data reside in subtrees of the Oracle Server owner's login home directory. The name of this directory is the value to which Oracle advises that you set the environment variable **ORACLE_BASE**.

Product Files Mature production sites require means to test a new version of Oracle software without impacting daily operations. Staging upgrades has been a difficult challenge for many Oracle sites because the focus of a default installation is that there is exactly one central repository of Oracle software. An inexperienced technician trying to complete an Oracle installation will rarely consider the requirements of an upgrade that isn't even planned yet. Building a structure to make upgrades easy is simple work, but it requires foresight and a complete understanding of the Oracle version switching mechanism discussed briefly in section 1.3.

> *Requirement 8.* It must be possible to execute multiple versions of applications software simultaneously. Cutover after upgrade must be as simple for the administrator and as transparent for the user community as possible.

Conforming to the following rule yields a structure that helps fulfill requirement 8:

OFA 4 *Store each version of Oracle Server distribution software in a directory matching the pattern* ***h/product/v****, where* ***h*** *is the login home directory of the Oracle software owner, and* ***v*** *represents the version of the software.*

Most OFA for UNIX implementations use values of ***v*** like **7.0.16**. Oracle Server patches involving changes only to version numbers right of the third decimal point (e.g., from 6.0.36.3 to 6.0.36.5) usually take place without elaborate staging, and thus most sites do not use values of ***v*** significant beyond the third decimal point.

An Oracle upgrade at an OFA compliant site then undertakes the following steps: (1) a new version directory is created in **product**; (2) the new Oracle Server version is installed in that directory; (3) a test database is created for confirmation of the new software; (4) after confirmation, the home directory column of **oratab** is updated for the row associated with the production instance; and (5) to cut over to the new version, all users exit and re-enter UNIX. Executing a fresh login (normally accomplished by leaving work one afternoon to return the next morning) causes the **oraenv** call in a user's **.profile**[4] to set the UNIX environment properly for operation with the newly validated software.

After an upgrade is judged successful, the next step becomes removal of the old version subtree, so that the space it consumes can be reclaimed. The safety with which the old version directory can be removed varies inversely

[4] . . . Or a **coraenv** call in a C shell user's **.cshrc**.

with the amount of custom material stored there. The Oracle administrator should ensure that the only files permitted to reside within the **product** subtree either are either copied or mechanically derived from the Oracle distribution media (such as executables created by linking distributed object files). Following this policy greatly simplifies upgrade cutover and cleanup by eliminating the need for intricate file-copy surgery to preserve original work. The lifespan of the **product/*v*** subtree matches the lifespan of a version of Oracle. Don't put files there if they will outlive version ***v*** of the software.

Administrative Files A key Oracle administrator skill is the ability to manage large amounts of data *about* an Oracle system. In the normal course of operation, for example, installers store programs that create databases; Oracle Server itself produces trace files; and administrators save structural records, instance parameters, performance statistics, backups, archives, and general logbook entries on each database. The volume of data to be administered increases as the number of databases on the system increases. These facts motivate the following requirement:

> *Requirement 9.* Administrative information about one database must be separated from that of others; there must be a reasonable structure for organization and storage of administrative data.

In exclusive mode environments,[5] conforming to the following rule fulfills requirement 9.

OFA 5 *For each database with **db_name**=**d**, store database administration files in the following subdirectories of **h/admin/d**:*

- **adhoc**—*ad hoc SQL scripts for a given database*
- **adump**—*audit trail trace files*
- **arch**—*archived redo log files*
- **bdump**—*background process trace files*
- **cdump**—*core dump files*
- **create**—*programs used to create the database*
- **exp**—*database export files*
- **logbook**—*files recording the status and history of the database*
- **pfile**—*instance parameter files*
- **udump**—*user SQL trace files*

*where **h** is the Oracle software owner's login home directory.*

Figure 5 shows the ***h*/admin/*d*** directories for three databases in a sample system; because space is limited, the picture does not show the **adhoc**, **adump**, etc. directories that reside beneath each named database directory.[6] The simple classification named here gives the administrator sufficiently many "file folders'' to store the necessary data about a given database. By keeping all original work pertinent to a specific database in the administrative subtree instead of in the product subtree, as so many people have implicitly done in the past by storing files in the **dbs** or **rdbms/admin** subtrees of **$ORACLE_HOME**, the administrator accomplishes the goal of keeping the product subtree free from files that must be carefully preserved at Oracle upgrade time.

Some administrative directories, such as **arch** and **exp**, are typically too large to store on the disk slice housing the Oracle owner's login home directory. These directories can be connected easily into the administrative subtree with symbolic links similar to the one shown earlier in Figure 3. Using symlinks gives a simple mechanism for storing a file anywhere you need without sacrificing the organization of your file system to physical size constraints.

Local Software UNIX is an especially open environment designed to enable addition of new capabilities into the operating system. General purpose administrative utilities like those listed in section 9 are typically executed from a directory created exclusively for site-specific utilities, such as **/var/opt/bin**. The OFA Standard similarly enables the Oracle administrator to add site-specific Oracle capabilities into the system in the **local** subtree of the Oracle owner's login

[5] For sites using Oracle Parallel Server, see section 5.

[6] The **TAR** directory shown in Figure 5 is created at some sites to enable site staff to record information about technical assistance requests (TARs) logged with Oracle Worldwide Support. The upper-case name distinguishes this directory from that of a database named 'tar,' if one were to exist.

home directory. During Core Technologies on-site engagements, for example, an Oracle consultant will populate this subtree with administrative utilities.

2.2 Oracle Database Files

Intuition tells most installers that Oracle database files should be separated from other files on a system. There are concrete reasons for doing so, among which: database files' lifespans differ from all other files on your system; and database files will require a different backup strategy than the other files on your system. A thoughtful naming strategy for database files eliminates a whole class of administrative problems. Experience yields the following requirement:

> *Requirement 10.* Database files should be named so that (a) database files are easily distinguishable from other files; (b) files of one database are easily distinguishable from files of another; (c) control files, redo log files, and data files are easily distinguishable from one another; and (d) the association of data file to tablespace is easily identifiable.

The following rule meets requirement 10:

OFA 6 *Name Oracle database files using the following patterns:*

- **/*pm*/*q*/*d*/control.ctl**—*control files*
- **/*pm*/*q*/*d*/redo*n*.log**—*redo log files*
- **/*pm*/*q*/*d*/*tn*.dbf**—*data files*

*where **pm** is a mount point name, **q** is a string denoting the separation of Oracle data from all other files, **d** is the **db_name** of the database, **n** is a distinguishing key that is fixed-length for a given file type, and **t** is an Oracle tablespace name. Never store any file other than a control, redo log, or data file associated with database **d** in **/pm/q/d**.*

In section 1.1 we discussed selection of mount point names, to which we refer here as **/*pm***. The two directory layers beneath the mount point level are valuable agents of organization. The ***q*** layer fulfills requirement 10.a. It is homologous to the **home** layer discussed in section 1.2, as it serves the same purpose of enabling an administrator to refer to a collection of I/O balanced files as a unit, in a zero maintenance way. Many people will choose ***q*=ORACLE**, as in the original OFA paper [Millsap 1993], or other obvious values like ***q*=oradata**. The ***d*** layer satisfies requirement 10.b. It eliminates confusion about the database association of a given file, because the name of the database will always be a component of the fully qualified path name of the file. For example, it is easy to see to which database **/u03/oradata/sab/system01.dbf** belongs.

Figure 6 is a graphical representation of a sample OFA compliant directory structure for storing I/O balanced database files that are independent of the Oracle software and administrative files. In this example, ***q*=oradata** on a system with three databases. The figure shows only the specific mount points **/u08**, **/u09**, and **/u10**, but the OFA administrator replicates this structure on every **/u[0-9][0-9]** mount point in the file system. It is a good idea to build the ***q*/*d*** structure for every database and every mount point, even if you do not intend to put database files on every mount point. The cost is only a few inodes,[7] and the benefit is that in an emergency, any free disk space on your system will be ready to house a file from any database that could require more space.

This structure makes it easy to distinguish Oracle database files of one database from Oracle database files of another. Using distinct suffixes for different types of database files makes it easy to distinguish one type of file from another, fulfilling requirement 10.c.

[7] The cost is $n(k+1)$ inodes, where n is the number of user data mount points on your system, and k is the number of databases on your system.

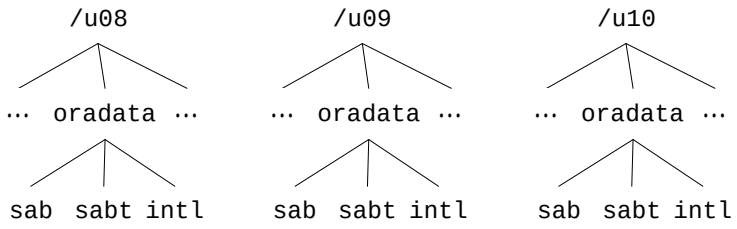

Figure 6. This figure shows the OFA structure for storing Oracle database files. Each mount point on the system contains a directory for holding Oracle data, **oradata** in this particular example. Other directories residing at this level might include subtrees named **home**, **app**, and so on. Beneath the Oracle data directory is a level separating the files of the various Oracle databases on the system. This example shows directories for three databases: **sab**, **sabt**, and **intl**. Note that in this structure, files from each database may be spread across as many disk drives as needed for I/O load balancing. Each mount point is equally well suited for storage of any database file.

Control Files Oracle control files contain structural information about the database, including relatively static data such as UNIX file names, and more dynamic data like the current redo log sequence number. For safety's sake, it is essential that the administrator create at least two control files on two different disks. Having three control file copies ensures that even if one file is lost, there remains a safe duplication. Because control file copies are always stored on different disks, they can have identical basenames; the ***/pm*** component distinguishes one control file from the others.

Redo Log Files Oracle redo log files record information necessary to roll forward a database in case of CPU or disk failure. Redo log files are written sequentially during transaction processing, and they are read only during archiving and recovery. Since redo log files for a database may all exist on a single drive, it is necessary to encode a distinguishing key into the basename of the file. This key ***n*** for redo log files is normally a three- or four-digit string denoting the redo log file's group and sequence within that group. For example, you might use a name like **redo0201.log** or **redo01b.log** to denote redo log member 1 of group 2.

Data Files Oracle data files are the physical manifestation of tablespaces. The OFA Standard meets requirement 10.d by using the tablespace name as the root of the data file's basename. Because two or more files from a given tablespace can reside within the same directory, it is necessary to encode a distinguishing key into the basename of each data file. The length of the key ***n*** is an almost universally accepted two. Numerals alone generate 100 unique keys in the range **00**, **01**, …, **99**. Most databases use far fewer than 100 files per tablespace. Common data file names are **system01.dbf**, **glx04.dbf**, and so on. However, if having 100 files per tablespace is not enough, then using two-digit alphanumeric lower-case keys from the set **[0-9a-z][0-9a-z]**, for example, yields $(10+26)^2$, or 1,296, unique values.

Personal Preferences Enthusiasts of the original OFA [Millsap 1993] may reel in algebraic anaphylaxis upon seeing the familiar **/u*n*/ORACLE/*d*** changed into the symbolic ***/pm/q/d***. The early OFA paper was designed to supply a simple, specific standard that installers could use straight out of the wrapper. However, when the material proved to be popular enough that administrators began to construct site standards around the OFA recommendations, the simple way of expressing a directory name recommendation proved to be inadequate.

Several rounds of discussion with our customers have made it clear that not all sites have exactly the same tastes in naming, and that the existence of two different preferences doesn't necessarily mean that someone is wrong. So there is no single correct answer to whether Oracle data directories are best named **ORACLE** or **oradata** or something else, and science

`/u[0-9][0-9]`	user-data directories
`/*/home/*`	user home directories
`/*/app/*`	user application software directories
`/*/app/applmgr`	Oracle apps software subtrees
`/*/app/oracle/product`	Oracle Server software subtrees
`/*/app/oracle/product/6.0.37`	Oracle Server 6.0.37 distribution files
`/*/app/oracle/admin/sab`	**sab** database administrative subtrees
`/*/app/oracle/admin/sab/arch/*`	**sab** database archived log files
`/*/oradata`	Oracle database directories
`/*/oradata/sab/*`	**sab** database files
`/*/oradata/sab/*.log`	**sab** database redo log files

Figure 7. These file name patterns are useful in an OFA compliant environment.

does not dictate whether database files are best kept four layers deep in the file tree or six. However, the *requirements* listed in this paper appear generally indisputable, and there seems to be unanimous agreement among thinkers about two assertions on storing Oracle data:

- The directory's actual name doesn't matter, as long as it is both (a) consistent with the names of other similar directories, and (b) chosen carefully to not misrepresent the contents of the directory.
- The level at which any type of I/O balanced files are stored doesn't matter, as long as it's the same level on every mount point.

The OFA Standard uses an algebraic-looking regular expression notation in the database file naming recommendation so that administrators are free to exploit the freedom of individual tastes, as long as the two underlying basis points expressed above are honored. Today, the site naming standards represented by the following names lie completely within the boundaries of OFA compliance:

```
/u01/ORACLE/sab/gld01.dbf
/disk04/oradata/pdnt/gld01.dbf
/db016/ora/mail/gld01.dbf
/mars/data/disk31/bnr1/gld01.dbf
/u08/app/oracle/data/pfin/gld01.dbf
```

Other standards are also appropriate in special circumstances. Responsible site administrators must consider the important points outlined in section 4 before deciding to implement Oracle on UNIX "raw devices.'' Section 6 addresses naming standards appropriate for consideration by very large database sites.

2.3 Exploiting the OFA Structure for Oracle Files

Figure 7 shows several useful UNIX patterns identifying classes of files that can be manipulated by a **find** command like the ones shown earlier in Figure 4. Figure 8 is a complete picture of the relationships among the Oracle for UNIX files in our familiar sample three-database system. In this example, the Oracle owner's login home directory is in **/u01/app**, and the files for this system's three databases are I/O balanced throughout subtrees named **/*/oradata**.

/	root mount point
u01/	'user data' mount point #1
app/	subtree for application software
oracle/	login home for the Oracle Server software owner
admin/	subtree for database administrative files
TAR/	subtree for Support logs
intl/	administrative subtree for 'intl' database
sab/	administrative subtree for 'sab' database
sabt/	administrative subtree for 'sabt' database
local/	subtree for local Oracle software
aps6/	an Oracle6 admin istrative software package
aps7/	an Oracle7 administrative software package
product/	Oracle Server distribution files
6.0.37/	**ORACLE_HOME** for 6.0.37 instances
7.0.16/	**ORACLE_HOME** for 7.0.16 instances
home/	subtree for login home directories
sbm/	home for a user
oradata/	subtree for Oracle data
intl/	subtree for 'intl' database files
sab/	subtree for 'sab' database files
sabt/	subtree for 'sabt' database files
u02/	'user data' mount point #2
app/	subtree for app software
applmgr/	home for the Oracle Applications owner
alr/	Oracle Alert
fnd/	Application Object Library
gl/ -> /u03/app/applmgr/gl	symbolic link to General Ledger files
...	more applications
home/	subtree for login home directories
mlm/	home for a user
oradata/	subtree for Oracle data
intl/	subtree for 'intl' database files
sab/	subtree for 'sab' database files
sabt/	subtree for 'sabt' database files
u03/	'user data' mount point #3
app/	subtree for applications software
applmgr/	auxiliary directory for Oracle Applications
gl/	Oracle General Ledger
home/	subtree for login home directories
vrm/	home for a user
oradata/	subtree for Oracle data
intl/	subtree for 'intl' database files
sab/	subtree for 'sab' database files
sabt/	subtree for 'sabt' database files
...	more mount points, etc.

Figure 8. This is a hierarchical directory listing for a sample OFA compliant system. Indentation shows the relationships among directories and the files and directories contained within.

3. Oracle Tablespaces

Tablespace is a name that was introduced in Oracle Version 6 for an entity that relates multiple database segments to multiple operating system files.

Segment Partitioning The tablespace forms the interface through which the logical database can be partitioned into different physical files in the operating system. Factors that affect decisions about separating Oracle segments into different tablespaces include:

- *Fragmentation character*. Dropping of segments causes tablespace free space fragmentation that can lead you to the doorstep of people who hope to "defragment'' your database in exchange for a few thousands of your dollars. Free space fragmentation is preventable. By separating segments with different lifespans into different tablespaces, the database creator prevents the problems associated with tablespace free space fragmentation.
- *I/O distribution*. Only at the tablespace level can the administrator determine what set of operating system files a segment will occupy. By separating segments with competing I/O requirements into different tablespaces, the database creator can make it possible to ensure well-balanced I/O loads across hardware components.
- *Administrative needs*. Only at the tablespace level can the administrator specify a collection of segments for backup, or restrict a user's privilege to consume database space. By separating segments with different administrative characteristics into different tablespaces, the database creator can make give the administrator an appropriate level of control over collections of segments.

The following requirement motivates decisions about tablespace creation:

> *Requirement 11*. Tablespace contents must be separated to (a) minimize tablespace free space fragmentation, (b) minimize I/O request contention; and (c) maximize administrative flexibility.

The next rule fulfills requirement 11.

OFA 7 *Separate groups of segments with different lifespans, I/O request demands, and backup frequencies among different tablespaces. For each Oracle database, create the following special tablespaces in addition to those needed for applications segments:*

- **SYSTEM**—*data dictionary segments only*
- **TEMP**—*temporary segments only*
- **RBS**—*rollback segments only*
- **TOOLS**—*general-purpose tools only*
- **USERS**—*miscellaneous user segments*

This standard has proven to have several tremendous benefits. For example, because dictionary segments are never dropped, and because no other segments that can be dropped are allowed in the **SYSTEM** tablespace, this scheme guarantees that the **SYSTEM** tablespace will never require a rebuild due to tablespace free space fragmentation. Because no rollback segment is stored in any tablespace holding applications data, the administrator is never blocked from taking an applications data tablespace offline for maintenance. Because segments are partitioned physically by type, the administrator can record and predict data volume growth rates without complicated tools.

Tablespace Names The OFA standard of embedding the name of a tablespace in the basename of its associated data files (OFA 6) means that UNIX file name length restrictions also restrict tablespace name lengths. Although Oracle tablespace names can be thirty characters long, portable UNIX file names are restricted to fourteen characters. Recall that the recommended standard for a data file basename is ***tn*.dbf**, where ***t*** is a tablespace name and ***n*** is a two-digit string. The six-character ***n*.dbf** suffix leaves eight characters remaining for ***t***. The recommended naming standard for tablespaces is thus:

OFA 8 *Name tablespaces connotatively with eight or fewer characters.*

Eight-character tablespace names not only simplify data file naming, they also make administrative reports about tablespaces much more "80-column friendly.'' The total number of tablespaces in a database is generally on the order of 100 or less, so inventing connotative names with eight characters is usually easy.

Connotation enables the administrator to divine the purpose of a tablespace by looking at its name. It is sometimes useful to encode information about what type of segment a tablespace is designed to store into the tablespace name. For example, the names **GLD** and **GLX** might connote that these tablespaces are designed to store Oracle General Ledger data and indexes, respectively. Figure 9 illustrates tablespace and file naming in an OFA compliant system.

File Type or Tablespace Name	File Name	Size (KB)
control	/u01/oradata/sab/control.ctl	
control	/u02/oradata/sab/control.ctl	
control	/u03/oradata/sab/control.ctl	
redo group 1	/u03/oradata/sab/redo0101.log	5,120
redo group 1	/u05/oradata/sab/redo0102.log	5,120
redo group 1	/u07/oradata/sab/redo0103.log	5,120
redo group 2	/u04/oradata/sab/redo0201.log	5,120
redo group 2	/u06/oradata/sab/redo0202.log	5,120
redo group 2	/u08/oradata/sab/redo0203.log	5,120
SYSTEM	/u02/oradata/sab/system01.dbf	65,536
TEMP	/u04/oradata/sab/temp01.dbf	131,072
RBS	/u03/oradata/sab/rbs01.dbf	65,536
TOOLS	/u07/oradata/sab/tools01.dbf	16,384
USERS	/u07/oradata/sab/users01.dbf	32,768
AOL	/u05/oradata/sab/aol01.dbf	98,304
GLD	/u02/oradata/sab/gld01.dbf	524,288
GLD	/u04/oradata/sab/gld02.dbf	524,288
GLX	/u05/oradata/sab/glx01.dbf	524,288

Figure 9. This is a file map of a sample OFA compliant database. Note the ease with which you can discern each file's mount point, its application, its database, and its tablespace. Note also that each file's type (control file, redo log file, or data file) is apparent by viewing its name.

Resist the temptation to embed reminders of the word *tablespace* in your tablespace names. Tablespaces are distinguishable as tablespaces by context, and their names do not need to convey information about their type. A competent Oracle administrator would not confuse a tablespace with another database object, so names like **TEMP_TABLESPACE** are patently unnecessary. Administrators learn as they gather experience that embedding type information into an object's name is usually a waste of motion.[8]

[8] People who, after the sermon, insist upon continuing to embed type information into the names of new things are of course free to do so. However, I am also free to fantasize that the grammar police will make these people use their own naming convention at home until they see how ridiculous it is. The parents of Billy the Kid, Attila the Hun, Winnie the Pooh, and Frosty the Snowman were evidently afflicted with such a curse.

4. Raw Devices *vs.* Buffered I/O

The term *raw device* is an informal synonym for *character special file*, which is an unmounted disk slice that Oracle can read and write without incurring the overhead of UNIX I/O buffering [Bach 1986, Leffler et al 1990]. A character special file has a fixed size because it is allocated an entire disk slice. Oracle Corporation uses raw devices extensively in TPC benchmarks to increase the number of transactions per second that Oracle can perform. As tempting as a potential performance gain sounds, use of raw I/O bears undeniable costs. The following sections identify some of the benefits and costs of using raw devices.

Benefits of Using Raw Devices The following factors weigh in favor of using raw devices:

- If you intend to use Oracle Parallel Server on a UNIX cluster, with multiple UNIX nodes manipulating a single database on a

shared disk subsystem, you must use raw devices. UNIX vendors have not yet implemented a way for nodes in a cluster to simultaneously mount a shared disk subsystem.

- Some platforms offer the capability for asynchronous I/O to boost output performance. Asynchronous I/O is normally available only with character special devices, outside the UNIX file system. Ask your hardware vendor or Oracle Worldwide Support for more details about your particular implementation.
- Using raw devices will in some cases increase throughput if either your processes frequently wait for I/O, or your system state CPU time is persistently excessive.[9]
- If you have variable disk partitioning,[10] using raw devices for redo log files is particularly attractive, because: Raw devices benefit you most for write-intensive, sequentially accessed data; and online redo log files are not included in normal operating system backup procedures.

Costs of Using Raw Devices The following factors weigh against using raw devices:

- Raw devices are more costly to administer than files in the UNIX file system. Operations that become more difficult are backup and recovery, I/O load balancing, and addition of files to the database. If you do not have the ability to thoroughly practice database recovery before taking your project live, then do not use raw devices. If you do not have sufficient disk volume that you can leave two or more large unformatted disk slices available for database growth or I/O balancing, then do not use raw devices.
- Raw I/O improves throughput in only a small percentage of production site situations. Applications in which I/O or I/O-related processing is not the bottleneck will not deliver better throughput by improving the performance of I/O and I/O-related processing. If I/O is your bottleneck, then before choosing to use raw devices, ensure that you have taken full advantage of performance optimization techniques that are less costly to you:
  - Carefully construct your application to minimize I/O.
  - Configure your operating system and instance parameters so that the Oracle Server operates as efficiently as possible.
  - Balance your I/O load across enough disks that each drive's data transfer rate is within the manufacturer's recommended limits for the drive and its controller.

  After these optimizations, you are likely to find that I/O and the CPU overhead associated with I/O is no longer your bottleneck, at which point moving to a raw device configuration will not improve throughput.
- If your UNIX implementation does not offer variable disk partitioning, a raw device Oracle configuration requires more disk volume than a similarly configured database using buffered I/O. Fixed partitioning constraints make it difficult to find enough suitably sized disk slices for redo logs and small data files. A dangerous consequence of any raw device configuration is that having only a few large slices tempts the inexperienced configuration planner to combine database segments that should be separated among Oracle tablespaces. The benefits of raw devices are not enough to overcome the user performance and server

[9] An interesting exception to the performance benefit traditionally attributed to raw device configurations occurs with applications that rely on table scans for data access. These applications actually perform better on mounted file systems than on raw devices. Improving the performance of an application that does frequent table scans begins with analysis of the application's data access methods, not with sweeping changes to the underlying operating system configuration.

[10] *Variable disk partitioning* is a capability offered by many UNIX vendors' logical volume management software. With this capability, an administrator can make any disk slice any size at all; without variable disk partitioning, an administrator must choose from a limited (usually small) number of ways to cut a disk into slices.

administrator workload costs of a poor fundamental configuration decision.

If yours is a big site with a sophisticated Oracle for UNIX administration staff with time for research, enough budget for several disk drives, and enough patience to leave some of your disk space unused but reserved for unanticipated growth and load balancing, then using raw devices may be economically feasible for you. If you use raw devices, you will need to meet the following requirement:

> *Requirement 12.* It must be possible to tune disk I/O load across all drives, including drives storing Oracle data in character special files.

The following rule enables the administrator to remedy a persistent I/O imbalance in a raw device configuration, because two character special files can trade places on disk only if they are precisely the same size.

OFA 9 *Choose a small set of standard sizes for all character special files that may be used to store Oracle database files.*

The Verdict Use of raw I/O does not necessarily increase a site's throughput, so a responsible administrator will study the issues before using them. Simply stated, if you are not running Oracle Parallel Server and testing shows there to be no performance improvement by using raw devices in your specific environment, then using them would be all cost and no gain for you. If testing shows raw devices to improve performance by an amount that outweighs the cost of administering them, then use raw devices. If you lack the time or expertise to invest in construction of a test to determine whether or not to use raw devices, then you probably also lack the time or expertise to administer them effectively.

When determining whether or not to use raw devices, do not ignore the possibility of using them only for some database files. Hybrid systems that use unbuffered I/O for sequential write-intensive files (e.g., redo log files in highly active OLTP environments) and buffered I/O for other files (e.g., control and data files that have to be backed up by a UNIX administrator) can offer the best mixture of low cost and high gain.

5. Oracle Parallel Server Administrative Files

Using Oracle Parallel Server (OPS) adds challenges to the database administrative task, because the OPS environment highlights the distinction between instance (a collection of processes and memory) and database (a collection of files). For example, assume that the **sab** database is simultaneously held open by two OPS instances on two UNIX nodes, **sab1** on node **node1** and **sab2** on node **node2**. Regardless of which instance hosts the connection, a report on a data dictionary table (or view, such as **dba_users**) will provide a consistent answer. However, a report produced on a dynamic performance table (or view, such as **v$parameter**) will give an answer wholly dependent upon which instance hosts the connection. Hence the motivation for the following requirement:

> *Requirement 13.* (a) Database specific administrative data must be stored in a central place accessible to all instance administrators; and (b) instance specific administrative data must be distinguishable as associated with a given instance by file name.

In other words, you must never have to search multiple directories to find all the reports on **dba_users** for a given database, and you must never have to look through a list of **v$parameter** reports to find the ones related to the instance you are inspecting.

We can refine our understanding of how to fulfill requirement 13 by observing the following features about OFA 5: the directories **arch**, **create**, and **exp** are database administrative directories; **adump**, **bdump**, **cdump**, **pfile**, and **udump** are instance administrative directories; and **adhoc** and **logbook** are curious mixtures of both.

/u01/app/oracle/admin/sab/	administrative directory for 'sab' database
adhoc/	directory for miscellaneous scripts
adump/	directory for audit file dumps
sab1/	audit file dest for 'sab1' instance
sab2/	audit file dest for 'sab2' instance
arch/	log archive dest for all instances
redo0001.arc	archived redo log file
bdump/	directory for background dump files
sab1/	background dump dest for 'sab1' instance
sab2/	background dump dest for 'sab2' instance
cdump/	directory for core dump files
sab1/	core dump dest for 'sab1' instance
sab2/	core dump dest for 'sab2' instance
create/	directory for creation scripts
exp/	directory for exports
930920full.dmp	Sep 20 full export dump file
export/	directory for export parfiles
import/	directory for import parfiles
logbook/	directory for 'sab' logbook entries
sab1/	directory for 'sab1' instance reports
params.lst	**v$parameter** report for 'sab1' instance
sab2/	directory for 'sab2' instance reports
params.lst	**v$parameter** report for 'sab2' instance
users.lst	**dba_users** report
pfile/	directory for instance parameter files
sab1/	directory for 'sab1' instance parameters
sab2/	directory for 'sab2' instance parameters
udump/	directory for user dump files
sab1/	user dump dest for 'sab1' instance
sab2/	user dump dest for 'sab2' instance

Figure 10. Administrative directory structure for dual instance OPS

One way of structuring the administrative directory for the **sab** database is depicted in Figure 10. In this example, each administrative subtree that must contain information specific to two or more instances uses another directory layer to denote the distinction in the file name. For example, consider the **admin/sab/logbook** directory. Operationally, an Oracle administrator would use this as the working directory for a SQL*Plus, SQL*DBA, or Server Manager session while administering the **sab** database. A data dictionary table report like **users.lst** would be spooled to the current working directory, regardless of whether the administrator is connected to **sab1** or **sab2**, to meet requirement 13. But a dynamic performance table report like **params.lst** would be spooled to the directory named for the instance to which the administrator running the report is connected. In addition to flexibly accommodating the storage of database and instance report history, the Figure 10 arrangement enables administrators to quickly find trace files created by a given instance. Therefore, the structure shown here fulfills requirement 13.b.

Further OPS environment complication derives from the fact that any instance can be administered from any node in the cluster. An administrator is equally likely to have logged into **node2** to administer the **sab** database as **node1**, and, through SQL*Net, the **node2** user can administer the **sab1** instance as **sab2**. However, to fulfill requirement 13.a, the administrator must use a single central administrative directory to store database reports. A mechanism for accommodating this require-

ment is to choose exactly one node to act as "administrative home'' for maintenance of a database. Each node housing an instance that connects to that database may have a distinct **product** directory, but the **admin/*d*** directory for a given database must be a remotely mounted link to the **admin/*d*** directory on the administrative home node. Thus, when an administrator logged onto **node2** sets **~oracle/admin/sab** as the session's current working directory, the working directory physically becomes the central **node1:/u01/app/oracle/admin/sab**, fulfilling requirement 13.

From this discussion, we can induce the following rule:

OFA 10 *If you are using Oracle Parallel Server, select exactly one node **N** to act as Oracle administrative home for the cluster to house the administrative subtree defined in rule OFA 5. Let **h** be the Oracle software owner's login home directory on node **N**. Create a directory named for each instance accessing database **d** within the **adump**, **bdump**, **cdump**, **logbook**, **pfile**, and **udump** directories of **N:h/admin/d**. On every node **n** in the cluster except **N**, mount the remote directory **N:h/admin/d** as the administrative directory for database **d** (i.e., as **n:h/admin/d**).*

6. Mount Points for VLDB Sites

There are at least two very different correct solutions to the mount point naming challenge, one for sites who can relax requirement 4 to satisfy requirement 2 economically, and one for sites able to afford massive disk farms. Most sites should use OFA 1. Only a handful of very large database (VLDB) sites worldwide should consider the following strategy.

OFA 11 *If you can afford enough hardware that:*

1. *You can guarantee that each disk drive[11] will contain database files from exactly one application; and*
2. *You can dedicate sufficiently many drives to each database to ensure that there will be no I/O bottleneck.*

*Then name mount points matching the pattern **/qdm** where **q** is a string denoting that Oracle data and nothing else is to be stored there, **d** is the value of the **db_name** init.ora parameter for the single database that will be stored there, and **m** is a unique fixed-length key that distinguishes one mount point for a given database from another.*

Mount point names like **/ora/intl01** connote a commitment to putting *only* control, redo log, and data files from the single Oracle database called "intl'' on a given disk slice.[12] If you adopt this standard, you commit never to need to use your Oracle data mount points for anything else. Only a very small percentage of all Oracle customers worldwide can make this commitment, and you should not adopt this particular VLDB component of the OFA Standard if there is a risk that you cannot.

7. Quick Reference Summary

The following sections are intended for use as a reference guide that summarizes the system requirements and OFA Standard recommendations discussed in this paper.

7.1 System Requirements

1. The file system must be organized so that it is easy to administer growth from: adding data into existing databases, adding users, creating databases, and adding hardware.
2. It must be possible to distribute I/O load across sufficiently many disk drives to prevent a performance bottleneck.
3. It may be necessary to minimize hardware cost.
4. It may be necessary to isolate the impact of drive failure across as few applications as possible.
5. It must be possible to distribute across two or more disk drives both (a) the collection of home directories and (b) the contents of an individual home directory.

[11] This is important: each disk *drive*, not disk *slice*. You violate requirement 4 if you place two applications on the same disk, even if they're stored in two different slices.

[12] In this example, ***q*=ora/**, ***d*=intl**, and ***m*=01**.

6. It must be possible to add or move login home directories without having to revise programs that refer to them.
7. Categories of files must be separated into independent directory subtrees so that files in one category are minimally affected by operations upon files in the other categories.
8. It must be possible to execute multiple versions of applications software simultaneously. Cutover after upgrade must be as simple for the administrator and as transparent for the user community as possible.
9. Administrative information about one database must be separated from that of others; there must be a reasonable structure for organization and storage of administrative data.
10. Database files should be named so that (a) database files are easily distinguishable from other files; (b) files of one database are easily distinguishable from files of another; (c) control files, redo log files, and data files are easily distinguishable from one another; and (d) the association of data file to tablespace is easily identifiable.
11. Tablespace contents must be separated to (a) minimize tablespace free space fragmentation, (b) minimize I/O request contention; and (c) maximize administrative flexibility.
12. It must be possible to tune disk I/O load across all drives, including drives storing Oracle data in character special files.
13. (a) Database specific administrative data must be stored in a central place accessible to all instance administrators; and (b) instance specific administrative data must be distinguishable as associated with a given instance by file name.

7.2 OFA Standard Recommendations

1. Name all mount points that will hold site-specific data to match the pattern ***/pm*** where ***p*** is a string constant chosen not to misrepresent the contents of any mount point, and ***m*** is a unique fixed-length key that distinguishes one mount point from another.
2. Name home directories matching the pattern ***/pm/h/u***, where ***pm*** is a mount point name, ***h*** is selected from a small set of standard directory names, and ***u*** is the name of the owner of the directory.
3. Refer to explicit path names only in files designed specifically to store them, such as the UNIX **/etc/passwd** file and the Oracle **oratab** file; refer to group memberships only in **/etc/group**.
4. Store each version of Oracle Server distribution software in a directory matching the pattern ***h*/product/*v***, where ***h*** is the login home directory of the Oracle software owner, and ***v*** represents the version of the software.
5. For each database with **db_name**=**d**, store database administration files in the following subdirectories of ***h*/admin/*d***:
   - **adhoc**—ad hoc SQL scripts for a given database
   - **adump**—audit trail trace files
   - **arch**—archived redo log files
   - **bdump**—background process trace files
   - **cdump**—core dump files
   - **create**—programs used to create the database
   - **exp**—database export files
   - **logbook**—files recording the status and history of the database
   - **pfile**—instance parameter files
   - **udump**—user SQL trace files

   where ***h*** is the Oracle software owner's login home directory.
6. Name Oracle database files using the following patterns:
   - ***/pm/q/d*/control.ctl**—control files
   - ***/pm/q/d*/redo*n*.log**—redo log files
   - ***/pm/q/d/tn*.dbf**—data files

   where ***pm*** is a mount point name, ***q*** is a string denoting the separation of Oracle data from all other files, ***d*** is the **db_name**

of the database, ***n*** is a distinguishing key that is fixed-length for a given file type, and ***t*** is an Oracle tablespace name. Never store any file other than a control, redo log, or data file associated with database ***d*** in ***/pm/q/d***.

7. Separate groups of segments with different lifespans, I/O request demands, and backup frequencies among different tablespaces. For each Oracle database, create the following special tablespaces in addition to those needed for applications segments:
   - **SYSTEM**—data dictionary segments only
   - **TEMP**—temporary segments only
   - **RBS**—rollback segments only
   - **TOOLS**—general-purpose tools only
   - **USERS**—miscellaneous user segments
8. Name tablespaces connotatively with eight or fewer characters.
9. Choose a small set of standard sizes for all character special files that may be used to store Oracle database files.
10. If you are using Oracle Parallel Server, select exactly one node ***N*** to act as Oracle administrative home for the cluster to house the administrative subtree defined in rule OFA 5. Let ***h*** be the Oracle software owner's login home directory on node ***N***. Create a directory named for each instance accessing database ***d*** within the **adump**, **bdump**, **cdump**, **logbook**, **pfile**, and **udump** directories of ***N*:*h*/admin/*d***. On every node ***n*** in the cluster except ***N***, mount the remote directory ***N*:*h*/admin/*d*** as the administrative directory for database ***d*** (i.e., as ***n*:*h*/admin/*d***).
11. If you can afford enough hardware that:
    1. You can guarantee that each disk drive will contain database files from exactly one application; and
    2. You can dedicate sufficiently many drives to each database to ensure that there will be no I/O bottleneck.

    Then name mount points matching the pattern ***/qdm*** where ***q*** is a string denoting that Oracle data and nothing else is to be stored there, ***d*** is the value of the **db_name** init.ora parameter for the single database that will be stored there, and ***m*** is a unique fixed-length key that distinguishes one mount point for a given database from another.

8. References

BACH, M. 1986. *The Design of the UNIX Operating System*. Englewood Cliffs, New Jersey: Prentice Hall.

FRISCH, A. 1991. *Essential System Administration*. Sebastopol, California: O'Reilly & Associates.

LEFFLER, S.; MCKUSICK, M.; KARELS, M.; QUARTERMAN, J. 1990. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Reading, Massachusetts: Addison-Wesley.

LOUKIDES, M. 1991. *System Performance Tuning*. Sebastopol, California: O'Reilly & Associates.

MILLSAP, C. 1993. "An optimal flexible architecture for a growing Oracle database." In *Oracle Magazine*, vol. VII no. 1 (Winter 1993): 41–46. Published originally as an Oracle Corporation white paper, 1990. Also published as "Configuring a growing Oracle V6 database for optimal performance" in 1991 *International Oracle User Week Proceedings*, paper 513.

NEMETH, E; SNYDER, G; SEEBASS, S. 1989. *UNIX System Administration Handbook*. Englewood Cliffs, New Jersey: Prentice Hall.

9. UNIX Utilities

UNIX site administrators are always on the lookout for useful utilities that ease the burden of administering complex applications. The pages at the end of this paper give you portable utilities that are typical of the small, reliable tools used by OFA compliant Oracle sites. Local general-purpose utilities such as these should normally be stored in the directory called **/var/opt/bin**. Enjoy.

LHD(L) **Oracle Services Utilities** **LHD(L)**

NAME

lhd — print login home directory name for a given user

SYNOPSIS

lhd [*login*]

DESCRIPTION

lhd prints the name of the login home directory for a given UNIX login, to allow the administrator to refer to a user's login home directory without hard-coding the path name. Using **`lhd *login*`** in the Bourne shell is the equivalent of ***~login*** in the C shell or KornShell. **lhd** enables creation of zero-maintenance administration programs that can survive file system changes without modification.

EXAMPLES

```
example$ . `lhd applmgr`/AOL.env
```

This example shows a line typical to that executed to set up a UNIX environment for Oracle Applications. The **.profile** containing this line of code would not require modification if the login home directory for **applmgr** were to change.

AUTHOR

Cary V. Millsap, Oracle Services

SOURCE

```
#!/bin/sh
#
# lhd - print login home directory name for a given user
#
# Cary Millsap, Oracle Services
# @(#)1.1 (93/05/17)

prog=`basename $0`
if [ $# -eq 0 ] ; then
        login=`whoami`
elif [ $# -eq 1 ] ; then
        login=$1
else
        echo "Usage: $prog login" >$2
        exit 2
fi
nawk -F: '$1==login {print $6}' login=$login /etc/passwd
```

GRPX(L) **Oracle Services Utilities** **GRPX(L)**

NAME

grpx — print the list of users belonging to a given group

SYNOPSIS

grpx *group*

DESCRIPTION

grpx prints the list of users belonging to a given group. **grpx** can be used to eliminate the need for hard-coding group memberships into administrative programs used to manipulate (e.g., back up, propagate files to) classes of users.

EXAMPLES

```
example$ for u in `grpx clerk` ; do
example>     cp /etc/skel/.profile `lhd $u`
example> done
```

This example shows how the administrator can propagate a skeleton **.profile** to the home directory for each member of a group. This code would not require modification if the membership list of the **clerk** group were to change.

AUTHOR

Cary V. Millsap, Oracle Services

SOURCE

```
#!/bin/sh
#
# grpx - print the list of users belonging to a given group
#
# Cary Millsap, Oracle Services
# @(#)1.1 (93/07/04)

prog=`basename $0`
if [ $# -ne 1 ] ; then
        echo "Usage: $prog group" >&2
        exit 2
fi
g=$1
# calculate group id of g
gid=`nawk -F: '$1==g {print $3}' g=$g /etc/group`
# list users whose default group id is gid
u1=`nawk -F: '$4==gid {print $1}' gid=$gid /etc/passwd`
# list users who are recorded members of g
u2=`nawk -F: '$1==g {gsub(/,/," "); print $4}' g=$g /etc/group`
# remove duplicates from the union of the two lists
echo $u1 $u2 | tr " " "\012" | sort | uniq | tr "\012" " "
echo
```