CONVEX NFS Concepts

Order No. DSW-109
First Edition
November 1990

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX NFS Concepts

First Edition
Order No. DSW-109

Printed in the United States of America

Revision Information for

CONVEX NFS Concepts

Edition	Document No.	Description
First	710-011730-000	Initial release with ConvexOS V9.0, November 1990.

Contents

List of Figures

Using This Book

Purpose and Audience

CONVEX NFS Concepts provides an introduction to the Network File System (NFS) and other facilities that work closely with NFS. Chapter one is an overview of NFS and its components. Chapter two discusses the Yellow Pages (YP) database service.

Chapter three explains the portmapper, a network communications service. Chapter four explains the network lock manager.

This book assumes no prior experience with NFS or with networking in general. It provides an introduction for someone who will configure and administer NFS ; typically this job belongs to the system manager for a CONVEX machine.

Technical Assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC).

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384.
- All other locations, contact the local CONVEX office.

Associated Documents

Using this software may require information not specific to the tasks described in this document.

For more information on the Convex NFS, you can order these books from CONVEX Computer Corporation:

- *CONVEX NFS System Manager's Guide* (DSW-113). This book explains administration of CONVEX NFS.
- *Convex NFS Reference* Set (DSW-111). This book provides a reference to the network programming facilities included with CONVEX NFS.
- *Convex NFS Programmer's Reference* (DSW-114). This book is the standard reference for CONVEX NFS.

Ordering Documentation

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, TX 75083-3851 USA

Include the order number or the exact title, as listed above.

Notational Conventions

This section discusses notational conventions used in this book.

General Conventions

In general, the following conventions are used in this guide:

- **`Bold constant-width font`** identifies user input in examples.
- *Italics*
  - Designate user-supplied variables in a command-line example.
  - Introduce new and important terms.
  - Identify variables in mathematical equations.
  - Indicate titles of documents.
- `Constant-width font` is used to designate input and output, including:
  - Command names and options.
  - System calls.
  - Data structures and types.
  - Directives, program statements, display examples, printout examples, and error messages returned.
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- Vertical ellipses show that lines of code have been left out of an example.
- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.
- The word "enter" in a phrase such as "enter **`ls`**" means that you type the command and then press **RETURN**.
- References to the *ConvexOS Programmer's Reference* appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Note

A Note highlights supplemental information.

Caution

A Caution highlights procedures or information necessary to avoid damage to equipment, software, or data.

Reader Response

If you have comments or questions about the contents of this book, please notify the CONVEX documentation department by using the Reader's Forum at the end of this document

CONVEX Network File System Overview

1

Basics

This chapter gives an overview of the CONVEX implementation of the Network File System (NFS), developed by Sun Microsystems. NFS allows users to mount directories across the network, and then to treat remote files as if they were local. Advanced users may want to skip the introductory sections and go to the working examples.

The Network File System (NFS) is a facility for sharing files in a heterogeneous environment of machines, operating systems, and networks. Sharing is accomplished by mounting a remote file system, then reading or writing files in place. NFS is open-ended and can be interfaced easily with other systems.

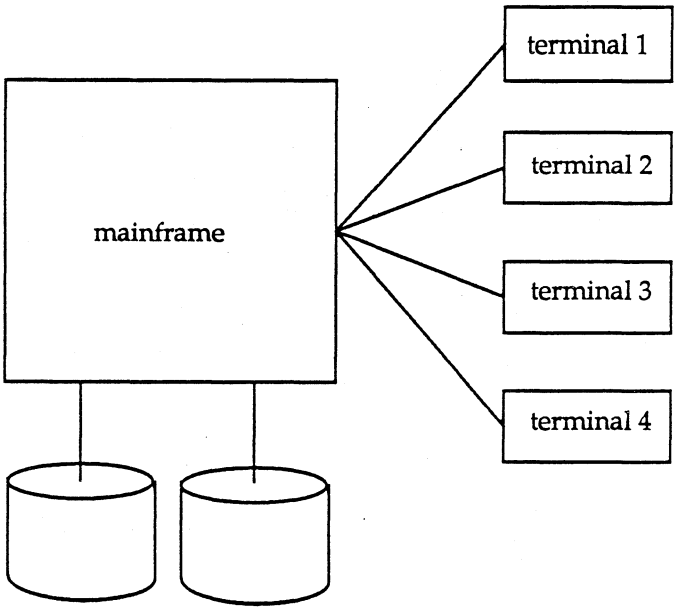
Even in a network environment, sharing programs and data is sometimes difficult. Files must be copied to each machine where they are needed, or users must log in to the remote machine where the required files are stored. Network logins are time-consuming, and having multiple copies of a file gets confusing when incompatible changes are made to separate copies.

To solve this problem, NFS provides a distributed file system that permits system A to access shared files on remote system B. Machine A requests resources provided by machine B, which makes the requested file systems available. Machine A mounts these file systems as local file systems. Thus, users can access remote files as if they were on the local machine.

NFS design does not extend the operating system onto the network. Instead, NFS fits into a network services architecture. Thus, NFS is not a distributed operating system, but rather, an interface that allows a variety of machines and operating systems to play the role of client or server.

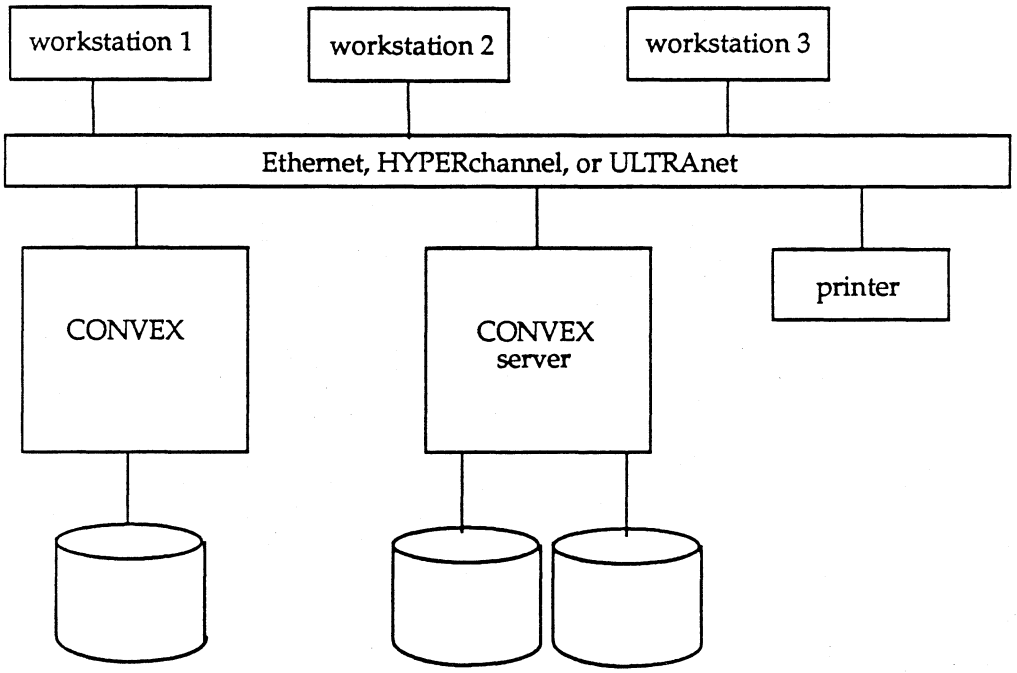
Computing Environments

The traditional computing environment looks like:



The major problem with this environment is competition for CPU cycles. The workstation environment solves that problem, but requires more disk drives.

A network environment looks like:



The goal of NFS is to make all disks available as needed. Individual workstations may have access to information residing anywhere on the network.

Terms and Concepts

The terms defined below appear throughout this manual:

- server - a machine that provides resources to the network.
- client - a machine that uses resources. A machine may be both a server and a client.
- user - person logged in on a machine.
- application - a program or set of programs that run on a machine.
- file system operations - the code implementing the operations of a file system.
- file system data - the data making up the file system's structure and contents.

A traditional file system is composed of directories and files, each of which has a corresponding inode (index node) containing administrative information about the file, such as location, size, ownership, permissions, and access times. Inodes are assigned unique numbers within a file system, but a file on one file system could have the same inode number as a file on another file system. This is a problem in a network environment, because remote file systems must be mounted dynamically, and numbering conflicts would cause havoc. The virtual file system (VFS) solves this problem. VFS is based on the vnode, an additional layer in the file system structure that permits multiple types of file systems.

Yellow Pages (YP) is a network service which eases the job of administering networked machines. YP is a centralized database. For a client on the network file system, this means that an application's access to data served by the YP is independent of the relative locations of the client and the server. The YP database on the server provides password, group, network, and host information to client machines. (For more information on YP, refer to Chapter 2.)

The Remote Procedure Call (RPC) facility provides a mechanism whereby a client process can have another process execute a procedure call, as if the caller process had executed the procedure call in its own address space (as in the local model of a procedure call). Because the caller and the server are now separate processes, they no longer have to live on the same machine.

The RPC mechanism is implemented as a library of procedures, and a specification for portable data transmission, known as the eXternal Data Representation (XDR). Both RPC and XDR are portable, providing a standard I/O library for interprocess communication.

The automounter mounts file systems on demand, based on pre-set conditions. Automounter function is transparent to users.

Design Goals

This section discusses some of the design features of NFS.

Transparent Information Access

Users can directly access files without knowing the network address of the data. To the user, there is no syntactical difference between reading or writing a file contained on a local disk, and reading or writing a file on a disk in the next building. Information on the network is truly distributed.

Machines and Operating Systems

No single vendor can supply tools for all the work that needs to get done, so appropriate services must be integrated on a network. NFS allows the exchange of data between different machines and operating systems.

Easy Extensibility

NFS allows integration of new software technologies without disturbing the extant software environment. NFS does not extend the underlying operating system onto the network, but, instead, offers a set of protocols for data exchange. These protocols can be easily extended.

Easy Network Administration

The administration of large networks can be complicated and time-consuming. With the network services provided with NFS, however, network administration should be no more difficult than administration of a set of local file systems on a time-sharing system.

YP is one example of a network service made possible with NFS. By storing password information and host addresses in a centralized database, YP eases the task of network administration. More information about the YP facility is presented in Chapter 2 of this book, in Chapter 3 of the *CONVEX NFS System Manager's Guide* and in the *CONVEX Yellow Pages Protocol Specification.*

Reliability

Reliability of the ConvexOS-based file system derives primarily from the robustness of the 4.2BSD file system. The file server protocol allows client workstations to continue operation even when the server crashes and reboots. Continuation after reboot is achieved without making assumptions about the fail-safe nature of the underlying server hardware.

The major advantage of a stateless server is robustness in the face of client, server, or network failures. Should a client fail, it is not necessary for a server (or system manager) to take action to continue normal operation. Should a server or the network fail, clients continue attempting to complete NFS operations until the server or network is fixed. This robustness is especially important in a complex network of heterogeneous systems, some of which may be running untested applications software or may be rebooted without warning.

Performance

The flexibility of NFS allows configuration for a variety of cost and performance trade-offs. For example, configuring servers with large, high-performance disks may yield better performance at lower cost than having many machines with small, inexpensive disks. Furthermore, it is possible to distribute the file system data across many servers and get the added benefit of multiprocessing without losing transparency. In the case of read-only files, you can keep copies on several servers to avoid bottlenecks, thanks to the `automount` utility.

NFS contains several performance enhancements, such as asynchronous service of multiple requests, caching of disk blocks, and asynchronous read-ahead and write-behind. The fact that caching and read-ahead occur on both client and server effectively increases the cache size and read-ahead distance. Caching and read-ahead are not required by the server; nothing is lost if cached information is thrown away. In the case of write-behind, both client and server attempt to flush critical information to disk, whenever necessary, to reduce the impact of an unanticipated failure; clients do not free write-behind blocks until the server verifies that the data is written.

NFS implementation provides extra functionality without a loss in local file system performance.

NFS Implementation

In the CONVEX implementation of NFS, three interfaces must be considered:

- the operating system interface
- the virtual file system (vfs) interface,
- the NFS interface.

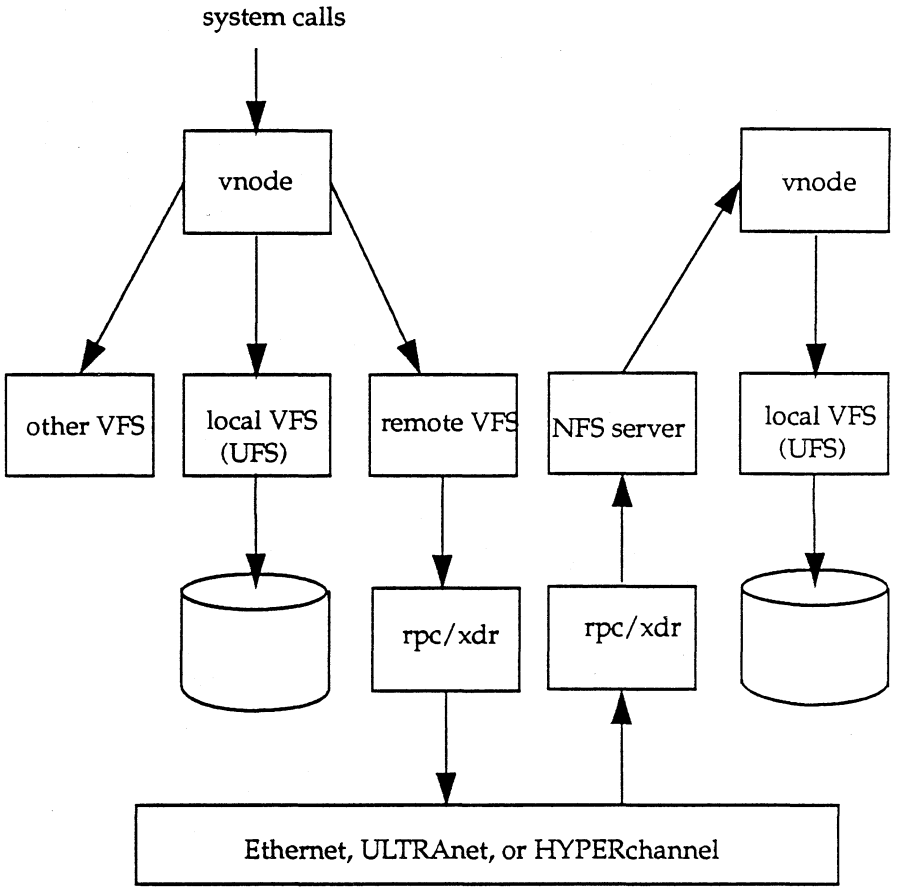
The operating system interface has been preserved in the CONVEX implementation of NFS, ensuring compatibility for existing applications.

Vnodes are a reimplementation of inodes that cleanly separate file system operations from the semantics of their implementation. Above the VFS interface, the operating system deals in vnodes; below this interface, the file system may or may not implement inodes. The VFS interface can connect the operating system to a variety of file systems (for example, 4.2BSD or MS-DOS). A local VFS connects to file system data on a local device.

The remote VFS defines and implements the NFS interface, using the remote procedure call (RPC) mechanism. RPC allows communication with remote services in a manner similar to the procedure calling mechanism available in many programming languages. RPC protocols are described using the eXternal Data Representation (XDR) package. XDR permits a machine-independent representation and definition of high-level protocols on the network.

Figure 1-1 illustrates the data flow through an NFS network.

Figure 1-1
Flow of request from a client (at top, left) to a collection of file systems.



In the case of access through a local VFS, requests are directed to file system data on devices connected to the client machine. In the case of access through a remote VFS, the request is passed through the RPC and XDR layers onto the net. In the current implementation, Convex uses UDP/IP protocols. On the server side, requests are passed through the RPC and XDR layers to an NFS server; the server uses vnodes to access one of its local VFSs and service the request. This path is retraced to return results.

The CONVEX implementation of NFS provides five types of transparency:

- file system Type - The vnode, in conjunction with one or more local VFSs (and possibly remote VFSs) permits an operating system (hence client and application) to interface transparently to a variety of file system types.
- file system Location - Since there is no differentiation between a local and a remote VFS, the location of file system data is transparent.
- Operating System Type - The RPC mechanism allows interconnection of a variety of operating systems on the network and makes the operating system type of a remote server transparent.
- Machine Type - The XDR definition facility allows a variety of machines to communicate on the network and makes the machine type of a remote server transparent.
- Network Type - RPC and XDR can be implemented for a variety of network and Internet protocols, thereby making the network type transparent.

Simpler NFS implementations are possible. In particular, a client (or server) may be added to the network by implementing one side of the NFS interface. An advantage of the CONVEX implementation is that the client and server sides are both provided to every machine; thus, it is possible for any machine to be client, server, or both. Users at client machines with disks can arrange to share over NFS without having to appeal to a system manager or configure a different system (if their machine is a workstation).

Interface

As mentioned in the preceding section, a major advantage of NFS is the ability to mix file systems. The sources to RPC and XDR are in the public domain and serve as a standard for anyone wishing to develop applications for the network. Furthermore, the NFS interface itself is open and can be used by anyone wishing to implement an NFS client or server for the network.

The NFS interface defines traditional file system operations for reading directories, creating and destroying files, reading and writing files, and reading and setting file attributes. File operations address files with an uninterpreted identifier, starting byte address, and length in bytes. Commands are provided for NFS servers to initiate service (`mountd` and `/etc/exportfs`) and to serve a portion of their file system to the network (`/etc/exports`). A client builds its view of the file systems available on the network with the `/etc/mount` command.

The NFS interface is defined so that a server can be stateless. This means that a server does not have to remember from one transaction to the next anything about its clients, transactions completed, or files affected. For example, there is no open operation, as this would imply state in the server. Of course, the interface uses an open operation, but the information in the operation is remembered by the client for use in later NFS operations.

An interesting problem occurs when an application unlinks an open file. This is done to achieve the effect of a temporary file that is automatically removed when the application terminates. If the file in question is served by NFS, the unlink removes the file, since the server does not remember that the file is open. Thus, subsequent operations on the file will fail. To avoid state on the server, the client operating system detects the situation, renames the file rather than unlinking it, and unlinks the file when the application terminates. In certain failure cases, this leaves unwanted "temporary" files on the server; the system manager should remove these files as a part of periodic file system maintenance. (See the *ConvexOS System Manager's Guide* for further information.)

Another example of how NFS provides an interface to ConvexOS without introducing state is the `/etc/mount` command. A client of NFS "builds" its view of the file system on its local devices using the `/etc/mount` command; thus, it is natural for the client to initiate contact with NFS and build its view of the file system on the network with an extended /etc/mount command. This `/etc/mount` command does not imply state in the server, since it only acquires information for the client to establish contact with a server. The `/etc/mount` command may be issued at any time, but is typically executed as a part of client initialization in the rc.local file. The corresponding `/etc/umount` command is only an informative message to the server, but it does change state in the client by modifying its view of the file system on the network.

An NFS server can be a client of another NFS server. A server will not, however, act as an intermediary between a client and another server. Instead, a client may ask what remote mounts the server has, then attempt to make similar remote mounts. The decision to disallow intermediary servers is based on several factors. First, an intermediary affects the performance characteristics of the system; the potential performance implications are so complex that it seems best to require direct communication between a client and server.

Second, an intermediary complicates access control; it is much simpler to require a client and server to establish direct agreements for service. NFS mounts can be established across gateways, but such mounts have no NFS intermediary between client and server.

NFS implements file protection by using authentication mechanisms built into RPC. This retains transparency for clients and applications that use file protection. Although the RPC definition allows other authentication schemes, their use may have adverse effects on transparency.

Although NFS is compatible with the ConvexOS file system, it does not support all ConvexOS file system operations. For example, the "special file" abstraction of devices is not supported for remote file systems because the interface to devices would greatly complicate the NFS interface; instead, devices are implemented in a local /dev VFS. Other incompatibilities are due to the fact that NFS servers are stateless. For example, file locking and guaranteed append mode are not supported in the remote case.

NFS, rcp, and ftp

NFS is composed of a modified kernel, a set of library routines, and a collection of utility commands; clients see a complete remote file system. NFS is an open system that can accommodate other machines on the network without compromising security.

Users may be familiar with two other networking facilities, `rcp` and `ftp`. The first is a remote copy utility program that uses the networking facilities of 4.2BSD to copy files from one machine to another. `ftp`, the user interface to the standard File Transfer Protocol, enables users to transfer files to and from a remote site.

The remote copy utility (`rcp`) allows data transfer only in units of files. The client of `rcp` supplies the path name of a file on a remote machine, and receives a stream of bytes in return. Access control is based on the client's login name and host name.

`rcp` is not transparent to the user, and it creates a redundant copy of the desired file. NFS, by contrast, is transparent—only one copy of the file is necessary. Also, `rcp` only copies files. In a sense, there needs to be one remote command for every regular command: for example, `rdiff` to perform differential file comparisons across machines. By providing entire file systems, NFS makes this unnecessary.

`ftp`, on the other hand, is an interactive program that provides many more features than `rcp`. `ftp` enables users to transfer files to and from a remote machine, delete files remotely, perform remote shell operations, and so forth. `ftp` presents many of the same problems encountered with `rcp`. `ftp` does not provide the transparent file access provided by NFS; neither does it eliminate the need for additional remote commands. NFS enables users to access remote file systems using essentially the same command set they use for local operations.

Example of NFS Use

In order for machines to share files via NFS, two things must happen. The server must export the files to be shared so that client machines can access them. The client machines must request that access by mounting the exported file systems. This section provides examples of these activities.

Mounting a Remote file system

Suppose that you want to read some online man pages. These pages are not available on the client machine, called xyz, but are available on a machine called docserv. Become superuser, then mount the directory containing the manuals as follows:

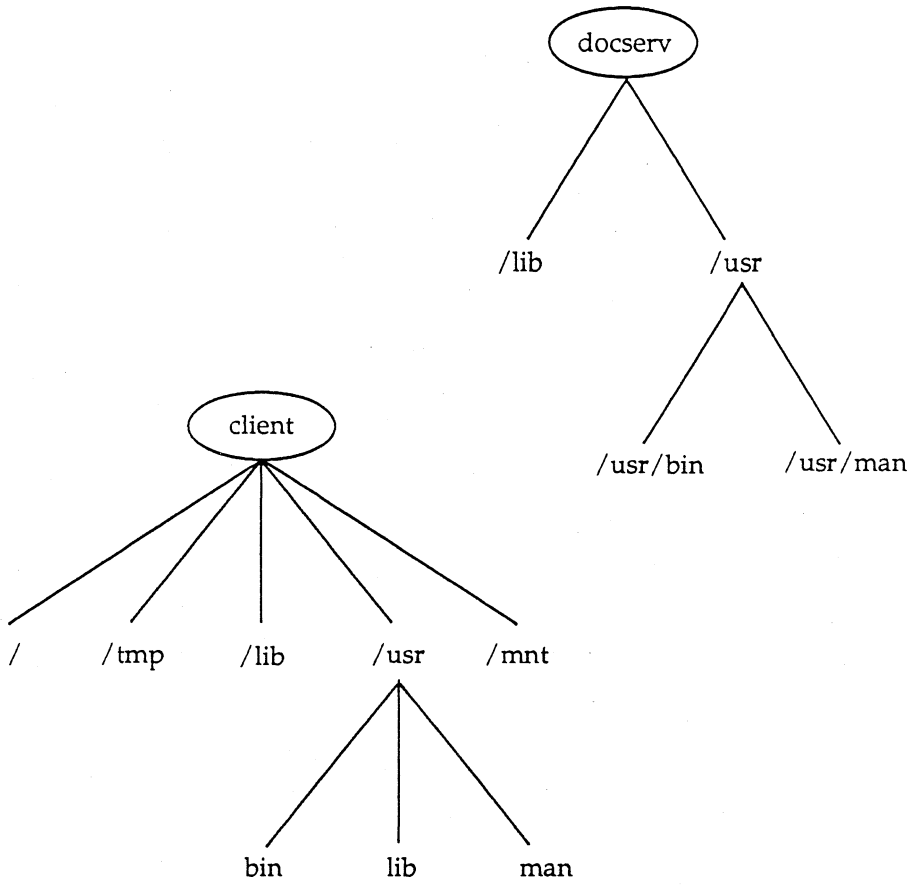
```
xyz# /etc/mount docserv:/usr/man /usr/man
```

After you export /usr/man from host docserv (see next example), you can use the `man` command whenever you want. Try running the `df` command after you have mounted the remote file system. Its output looks something like this:

```
file system          kbytes    used    avail   capacityMounted on
/dev/da0a            4775      2765    1532    64%      /
/dev/da0e            5695      3666    1459    72%      /mnt
/dev/st8             81567     2908    70502   4%       /tmp
/dev/st14            203391    39655   143396  22%      /doc
docserv:/usr/man346111     216894  94605   70%      /usr/man
```

You can remotely mount not only file systems, but also directory hierarchies inside file systems. In this example, /usr/man is not a real mount point, it is a subdirectory of the /usr file system. Figure 1-2 is a diagram of the file systems described above. Ellipses represent machines and boxes represent remote file systems.

Figure 1-2
Mounting Directories



Exporting a file system

To export the directory /usr/man from machine docserv, do the following:

1. Become superuser.
2. Edit the file /etc/exports. If you want the files to be available from machine xyz, then add this line to /etc/exports:
   ```
   /usr/man     -access=xyz
   ```
   Without the `access` option, anybody on the network can remotely mount the directory /usr/man.
3. After you have edited the file, run **`/usr/etc/exportfs -a`** to update the list of exported file systems inside the kernel.

Now your colleague can remotely mount the source directory by issuing the command:

```
xyz# /etc/mount docserv:/usr/man /usr/man
```

Because both you and users on xyz can change files on /usr/man, it is best to use a source code control system for concurrency control.

You can enable or disable over-the-network superuser privileges for client machines on either a machine-by-machine basis or a file system-by-file system basis. For details, see "Superuser Access to Remote Files" in the *CONVEX NFS System Manager's Guide.*

Administering a Server Machine

System administrators must know how to set up the NFS server so that client workstations can mount all necessary file systems. You export file systems (that is, make them available) by placing appropriate lines in the /etc/exports file. Here is a sample /etc/exports file for a typical server machine:

```
/
/mnt
/mnt2
/mnt/src  -access=staff
```

A netgroup, such as staff, may be specified after the file system, in which case remote mounts are limited to machines that are members of this netgroup. At any time, the system manager can see which file systems have been mounted remotely, by executing the /usr/etc/showmount command.

The Yellow Pages

2

Yellow Pages (YP) permit password information and host addresses for an entire network to be held in a single database comprised of the various YP maps. This greatly eases the task of system and network administration.

The most obvious use of YP is for administration of /etc/passwd. Because NFS uses a protection scheme across the network, it is advantageous to have a common /etc/passwd database for all machines on the network. YP allows a single point of administration and gives all machines access to a recent version of the data. Installing the YP version of /etc/passwd does not require changes to existing applications; they are simply relinked with library routines that know about YP service. (This is one reason that UIDs should be constant for all NFS and YP server clients.)

Conventions have been added to library routines that access /etc/passwd to allow each client to administer its own local subset of /etc/passwd; the local subset modifies the client's view of the system version. Thus, a client can accomplish a small amount of personalization without requiring assistance from the system manager.

The YP interface is implemented using RPC and XDR, so the service is available to a variety of operating systems and machine types. Because YP servers do not interpret data, it is possible for new databases to take advantage of the YP service without modifying the servers.

What Is YP?

YP constitutes a distributed network lookup service:

- YP is a lookup service; it maintains a set of databases for querying. Programs can ask for the value associated with a particular key, or all keys, in a database.
- YP is a network service; programs need not know the location of data or how it is stored. Instead, they use a network protocol to communicate with a YP server that knows those details.
- YP is distributed; databases may be fully replicated on several machines known as YP servers. Servers propagate updated databases among themselves, thus ensuring consistency. At steady state, it doesn't matter which server answers a request; the answer is the same everywhere.

YP Maps

YP serves information stored in YP maps. Each map contains a set of keys and associated values. For example, the hosts map contains (as keys) all host names on a network and (as values) the corresponding Internet addresses. Each YP map has a map name, used by programs to access data in the map. Programs must know the format of data in the map. Most maps are derived from ASCII files formerly found in the /etc directory: passwd, group, hosts, networks, and others. The format of data in the YP map is usually identical to the format of the ASCII file. Maps are implemented by `dbm` files located in /etc/yp subdirectories on YP servers.

YP Domain

A YP domain is a named set of YP maps. You can determine your YP domain by executing the `domainname` command. YP domains are different from Internet domains and sendmail domains. A YP domain is a directory in /etc/yp containing a set of maps. For example, your system might be in the following domains:

- YP domain dbproj
- Internet domain dept.comp.COM
- sendmail domain dept.comp.COM

A domain name is required for retrieving data from a YP database. For example, if your YP domain is "convex" and you want to find the Internet address of host "dbserver", you must ask YP for the value associated with the key "dbserver" in the map "hosts.byname" within the YP domain "convex".

Each machine on the network belongs to a default domain, set in /etc/rc.local at boot time with the `domainname` command.

A YP server holds all the maps of a YP domain in a subdirectory of /etc/yp, named after the domain. In the example above, maps for the convex domain would be held in /etc/yp/convex. This information is used internally by YP.

YP Servers and Clients

Servers and clients are defined differently for YP than they are for NFS. YP servers provide resources, and YP clients consume these resources. A server or a client is not necessarily the same as a machine. To illustrate, let's consider two different services: NFS and YP.

- NFS allows client machines to mount remote filesystems and access files in place, provided a server machine has exported the filesystem. However, a server that exports filesystems may also mount exported filesystems, thus becoming a client. So a given machine may be both server and client, or client only, or server only.

- The YP server, by contrast, is a process (rather than a machine) running on a machine that may be an NFS server. A process, such as `ls`, can request information out of the YP database, removing the need to have such information on every machine. All processes that use YP services are YP clients. Sometimes clients are served by YP servers on the same machine, but other times by YP servers running on another machine. If a remote machine running a YP server process crashes, client processes can obtain YP services from another machine. Thus, YP services are almost always available.

YP Masters and Slaves

YP servers are either master or slave. For any map, one YP server is designated the master; all changes to the map should be made on that machine. The changes propagate from master to slaves. A newly built map is time-stamped internally when `makedbm` creates it. If you build a YP map on a slave server, you break the YP update algorithm (temporarily), and must synchronize versions manually. Therefore, after you decide which server is the master, do all database updates and builds there, not on slaves.

It is possible for different maps to have different servers as master. A server may even be master with regard to one map, and slave with regard to another. This can get confusing quickly. It is recommended that a single server be master for all maps created by `ypinit` in a single domain. This document assumes the simple case, in which one server is master for all maps in a database.

YP Organization

YP is a network service containing network-wide databases such as /etc/hosts. Servers spread throughout the network contain copies of the databases. When an arbitrary machine on the network wants to look up something in /etc/hosts, it makes an RPC call to one of the servers to get the information. One server is the master—the only one whose database may be modified. The other servers are slaves, and they are periodically updated so that their information is synchronized with that of the master.

YP can serve any number of databases. Normally, that includes files that previously lived in /etc, such as /etc/hosts and /etc/networks. However, users can add their own databases to YP.

YP itself simply serves information and has no idea what it means. Thus, two parts of YP must be considered: how it operates, and what files formerly in /etc now live in YP. These location changes affect many users the first time YP is installed.

YP Operation

The following sections explain how YP handles names and data. Requirements for YP servers and clients are also described.

Naming

Imagine a company with two different networks, each of which has its own separate list of hosts and passwords. Within each network, user names, numerical user IDs, and host names are unique. There is, however, duplication between the two networks. If these two networks are ever connected, chaos could result. The host name, returned by the `hostname` command and the `gethostname` system call, may no longer uniquely identify a machine. Thus a new command and system call, `domainname` and `getdomainname`, have been added. The two networks just mentioned could be given different domain names; it is, however, always simpler to use a single domain whenever possible.

The relevance of domains to YP is that data is stored in /etc/yp/domainname. In particular, a machine can contain data for several different domains.

Data Storage

The data is stored in dbm format. Thus, the database hosts.byname for the domain convex is stored as /etc/yp/convex/hosts.byname.pag and /etc/yp/convex/hosts.byname.dir. The command makedbm takes an ASCII file such as /etc/hosts and converts it into a dbm file suitable for use by the YP. However, system managers normally use the makefile in /etc/yp to create new dbm files. This makefile calls makedbm.

Servers

To become a server, a machine must contain YP databases and must be running the YP daemon ypserv. The ypinit command invokes this daemon automatically. It also takes a flag saying whether you are creating a master or a slave. When you update the master copy of a database, the changes are propagated to the slave servers. Or, you can force the change to be propagated to all the slaves with the yppush command. Conversely, from a slave, the ypxfr command gets the latest information from the master. The makefile in /etc/yp first executes makedbm to make a new database, then calls yppush to propagate the change throughout the network.

Clients

Remember that a client machine (which is not a server) does not access local copies of /etc files, but makes an RPC call to a YP server each time it needs information from a YP database. The ypbind daemon remembers the name of a server. When a client boots, ypbind broadcasts asking for the name of the YP server. Similarly, ypbind broadcasts, asking for the name of a new YP server if the old server crashes. The ypwhich command gives the name of the server that ypbind currently remembers.

Because client machines no longer have entire copies of files in the YP, two new commands ypcat and ypmatch have been provided. The command ypcat hosts is equivalent to rsh server cat /etc/hosts; ypcat passwd is equivalent to rsh server cat /etc/passwd. Searching through the password file no longer suffices to look for someone's password entry. You must issue one of the following commands:

```
% ypcat passwd | grep username
% ypmatch username passwd
```

where username is the login name for which you are searching.

Changing Passwords

To change data in the YP, the system manager must log in to the master machine and edit databases there; ypwhich tells where the master server is. Because changing a password is a common operation, however, the yppasswd command has been provided to change your YP password. It has the same user interface as the passwd command. This command works only if the yppasswd server has been started on the YP master server machine.

Default YP Files

By default, CONVEX systems have several files from /etc in the YP:

```
/ethers
/group
/hosts
/netgroup
/networks
/passwd
/protocols
```

```
/pwrestrict
/services
```

Library routines such as

```
getgrent(3)
gethostent(3)
getpwent(3)
getpwrestent(3)
```

have been rewritten to take advantage of the YP. When YP is first installed, C programs that call these library routines must be relinked in order to function correctly.

hosts file

The hosts file is stored as two different files in the YP. The first, hosts.byname, is indexed by hostname. The second, hosts.byaddr, is indexed by Internet address. This file expands into four files, with suffixes .pag, and .dir. When a user program calls the library routine `gethostbyname`, a single RPC call to a server retrieves the entry from the hosts.byname file. Similarly, `gethostbyaddr` retrieves the entry from the hosts.byaddr file. Of course, if the YP is not running (which is caused by commenting `ypbind` out of the /etc/rc file), `gethostbyname` reads the /etc/hosts files.

➜ Note

The hosts file configuration explained above does not apply if the nameserver `named` is running. See the named(8) man page for the configuration required by the nameserver.

Maps sometimes have nicknames. Although the `ypcat` command is a general YP database print program, it knows about the standard files in the YP. Thus, `ypcat` hosts file is translated into `ypcat` hosts.byaddr, since there is no file called hosts in the YP. The command

```
ypcat -x
```

furnishes a list of expanded nicknames.

Normally, the hosts file for the YP is the same as the /etc/hosts file on the machine serving as a YP master. In this case, when you run make the makefile in /etc/yp checks to see if /etc/hosts is newer than the dbm file. If it is, it uses a `sed` script to recreate hosts.byname and hosts.byaddr, run them through `makedbm`, and then call `yppush`. See ypmake(8) for details.

passwd file

The passwd file is similar to the hosts file. It exists as two separate files: passwd.byname and passwd.byuid. The `ypcat` program prints it, and `ypmake` updates it. However, if `getpwent` always went directly to the YP, as does `gethostent`, everyone would be forced to have an identical password file. Consequently, `getpwent` reads the local /etc/passwd file. It interprets "+" entries in the password file to mean "interpolate entries from the YP database." If you wrote a simple program using `getpwent` to print all the entries from your password file, it would print a virtual password file; rather than printing out + signs, it would print whatever entries the local password file included from the YP database.

Others

The following files

- /etc/group
- /etc/hosts
- /etc/passwd
- /etc/pwrestrict

us the same + sign notation as /etc/passwd. `getgrent` and `getpwrestent` consult the YP only if explicitly told to do so by the /etc/group and /etc/pwrestrict files.

The following files,

- /etc/networks
- /etc/services
- /etc/protocols
- /etc/ethers
- /etc/netgroup

do not use the + sign notation and are treated like /etc/hosts. For these files, the library routines go directly to the YP without consulting the local files.

The Portmapper 3

Client programs need a way to find server programs: they need a way to look up and find the port numbers of server programs. (A port is a logical communications channel in a host — by waiting on a port, a process receives messages from the network). Naming of services by way of the port-number segment of their IP address is mandated by Internet protocols. Therefore, clients must determine which ports are associated with services they wish to use.

Network transport services do not provide such a service; they merely provide process-to-process message transfer across a network. A message typically contains a transport address that contains a network number, a host number, and a port number.

How a process waits on a port varies on operating system implementation, but all provide mechanisms that suspend a process until a message arrives at a port. Thus, messages are not sent across networks to receiving processes, but rather to the ports at which receiving processes wait for messages. Ports allow message receivers to be specified in a way that is independent of the conventions of the receiving operating system.

The portmapper protocol defines a network service that provides a standard way for clients to look up the port number of any remote program supported by a server. Because the portmapper protocol can be implemented on any transport that provides the equivalent of ports, it provides a single solution to a general problem that works for all clients, all servers, and all networks.

Port Registration

Every portmapper on every host is associated with port number 111. The portmapper is the only NFS network service that must have a well-known dedicated port. Other network services can be assigned port numbers statically or dynamically if those services register their ports with their host's portmapper. For example, a server program based on the RPC library typically gets a port number at run time by calling an RPC library procedure. A network service can be associated with port number 256 on one server and with port number 885 on another. On a given host, a service can be associated with a different port every time its server program is started.

Delegating port-to-remote program mapping to portmappers also automates port number administration. Statically mapping ports and remote programs in a file duplicated on each client would require updating all mapping files whenever a new remote program was introduced to a network.

Portmap

The port-to-program mappings that are maintained by the portmapper server are called a portmap. The portmapper is started automatically from /etc/rc.local when a machine is booted. Both server programs and client programs call portmapper procedures.

Although client and server programs and server machines are usually distinct, they need not be. A server program can also be a client program, as when an NFS server calls a portmapper server. Likewise, when a client program directs a "remote" procedure call to its own machine, the machine acts as both client and server.

As part of its initialization, a server program calls its host's portmapper to create a portmap entry. Whereas server programs call portmappers to update portmap entries, clients call portmappers to query portmap entries. To find a remote program's port, a client sends an RPC call message to a server's portmapper; if the remote program is supported on the server, the portmapper returns the relevant port number in an RPC reply message. The client program can then send RPC call messages to the remote program's port. A client program can minimize its portmapper calls by caching the port numbers of recently called remote programs.

The portmapper provides an inherently stateful service because a portmap is a set of associations between registrants and ports.

The portmapper protocol provides a procedure, `callit`, by which the portmapper can assist a client in making a remote procedure call. A client program passes the target procedure's program number, looks up the target procedure's port number in the portmap, and sends an RPC call message to the target procedure, including in it the arguments received from the client. When the target procedure returns results to `callit`, `callit` returns the results to the client program; the target procedure's port number is also returned so the client can subsequently call the target procedure directly.

A client has no way to broadcast a remote procedure call directly, because every instance of a remote program can be mapped to a different port on every server. The portmapper `callit` procedure can be used to broadcast a remote procedure call indirectly, since all portmappers are associated with port number 111. One way for a client to find a server running a remote program is to broadcast a call to `callit` asking it to call procedure 0 of the desired remote program. If this call is broadcast to all servers, the first reply received is likely to be from the server with the lightest workload.

The RPC library provides an interface to all portmapper procedures. Some of the RPC library procedures also call portmappers automatically on behalf of client and server programs.

The Network Lock Manager

4

Locking prevents multiple processes from modifying the same file at the same time and allows cooperating processes to synchronize access to shared files. The user interfaces with the network locking service using the `lockf` system-call interface and rarely requires any detailed knowledge of how it works.

Having the file system spread across multiple machines is not a complication—unless a crash occurs. In an NFS environment, where multiple machines can have access to the same file at the same time, the process of recovering from a crash is necessarily more complex than in a non-network environment. Furthermore, locking is inherently stateful.

If a server crashes, clients with locked files must be able to recover their locks. If a client crashes, its servers must hold the client's locks while it recovers. To preserve NFS's overall transparency, the recovery of lost locks must not require the intervention of the applications themselves. This is accomplished as follows:

- Basic file access operations, such as read and write, use the stateless NFS protocol. All interactions between NFS servers and clients are atomic—the server does not remember anything about its clients from one interaction to the next. In the case of a server crash, client applications simply sleep until the server comes back up and their NFS operations can complete.

- Stateful services (those that require the server to maintain client information from one transaction to the next), such as the locking service, are not part of NFS. They are separate services that use the status monitor. There are two specific state-related problems involved in providing locking in a network context:

  — If the client has crashed, the server can hold the lock forever
  — If the server has crashed, it loses its state (including all its lock information) when it recovers.

The Network Lock Manager solves both of these problems by cooperating with the Network Status Monitor to ensure that the lock manager is notified of relevant machine crashes. Its own protocol allows it to recover the lock information it needs when crashed machines recover.

The lock manager (`lockd`) and the status monitor (`statd`) are both network-service daemons. They run at user level, but are essential to the kernel's ability to provide fundamental network services, and therefore run on all network machines. Like other network-service daemons, which provide, for example, remote-execution services (`rexd`) and remote-login services (`rlogind`), they are extensions to the kernel which, for reasons of space, efficiency, and organization, are implemented as daemons.

Application programs that need a network service can either call the appropriate daemon directly with RPC/XDR or use a system call such as `lockf`, to call the kernel. In this later case, the kernel uses RPC to call the daemon. The network daemons communicate among themselves with RPC (refer to "The Locking Protocol" section for some details of the lock manager protocol). The daemon-based approach to network services allows users to customize services. For example, it is possible for users to alter the lock manager to provide locking in a different style.

At each server site, a lock manager process accepts lock requests, made on behalf of client processes by a remote lock manager or made on behalf of local processes by the kernel. The client and server lock managers communicate using RPC calls. Upon receiving a remote lock request for a machine on which it holds no locks, the lock manager registers its interest in that machine with the local status monitor and waits for the monitor to notify it the machine is up. The monitor continues to watch the status of registered machines, and notifies the lock manager when one of them is rebooted (after a crash). If the lock request is for a local file, the lock manager tries to satisfy it and communicates back to the application along the appropriate RPC path.

The crash recovery procedure is very simple. If the failure of a client is detected, the server releases the failed client's locks, on the assumption that the client application will request locks again as needed. If the recovery of a server is detected, the client lock manager re-transmits all lock requests previously granted by the recovered server. The server uses this re-transmitted information to reconstruct its locking state. See "The Locking Protocol" below for more details.

The locking service is essentially stateless. To be more precise, its state information is carefully circumscribed within a pair of system daemons that are set up for automatic, application-transparent crash recovery. If a server crashes, thus losing its state, it expects that its clients will be notified of the crash and send it the information that it needs to reconstruct its state. The key in this approach is the status monitor, which the lock manager uses to detect both client and server failures.

The Locking Protocol

The lock style implemented by the network lock manager is that specified in the *AT&T System V Interface Definition.* (Refer to the lockf(3) and fcntl(2) man pages for details.) There is no interaction between the `lockd`'s locks and `flock` locks, which remain supported, but which should be used for non-network applications only.

Locks are advisory only because cooperating processes can do whatever they wish without mandatory locks. In addition, mandatory locks pose serious security problems—if /etc/passwd is locked against reading, the whole system freezes. There are four basic kernel to lock manager requests:

- KLM_LOCK - Lock the specified record.
- KLM_UNLOCK - Unlock the specified record.
- KLM_TEST - Test if the specified record is locked.
- KLM_CANCEL - Cancel an outstanding lock request.

Despite the fact that the network lock manager adheres to the `lock` and `fcntl` semantics, there are a few subtle points about its behavior that deserve mention. These arise directly from the nature of the network:

- When an NFS client goes down, lock managers on all of its servers are notified by their status monitors, and they simply release its locks, because it will request them again when it wants them. When a server crashes, however, the clients wait for it to come back up. When it does, its lock manager gives the client lock managers a grace period to submit lock reclaim requests, and during this period accepts only reclaim requests. The client status monitors notify their respective lock managers when the server recovers. The default grace period is 45 seconds.

- It is possible that, after a server crash, a client will not be able to recover a lock that it had on a file on that server. This can happen if another process beats the recovering application process to the lock. In this case, the SIGLOST signal is sent to the process (the default action for this signal is to kill the application).

- The local client lock manager does not reply to the kernel lock request until the server lock manager has responded to it. Furthermore, if the lock request is on a server new to the local lock manager, the lock manager registers its interest in that server with the local status monitor and waits for its reply. Thus, if either the status monitor or the server's lock manager are unavailable, the reply to a lock request for remote data is delayed until it becomes available.

Network Status Monitor

The Network Status Monitor is explained alongside the lock manager because the lock manager relies heavily on the status monitor to maintain the inherently stateful locking service within the stateless NFS environment. However, the status monitor is very general and can also be used to support other kinds of stateful network services and applications. Normally, crash recovery is one of the most difficult aspects of network application development and requires a major design and installation effort. The status monitor greatly simplifies this task.

The status monitor provides a general framework for collecting network status information. A daemon that runs on all network machines, it implements a simple protocol that allows applications to easily monitor the status of other machines. Its use improves overall robustness and avoids situations in which applications running on different machines (or even on the same machine) disagree about the status of a site—a potentially dangerous situation that can lead to inconsistencies in many applications.

Applications register machines that they are interested in with the status monitor. The monitor tracks the status of those machines. When a machine recovers from a crash, it notifies the interested applications to that effect; the applications then take whatever actions are necessary to reestablish a consistent state.

There are several major advantages to this approach:

- Only applications that use stateful services incur the overhead—in time and in code—of dealing with the status monitor.

- Implementation of stateful network applications is easier because the status monitor shields application developers from the complexity of the network.

Reporting Problems

A

This appendix introduces the CONVEX Technical Assistance Center (TAC) and the `contact` utility.

The `contact` utility is an online system for reporting problems to the TAC. To use it, enter **`contact`** at the system prompt and answer the questions as they appear on the screen.

This appendix describes:

- Prerequisites for using `contact`
- Tips for using `contact`
- The step-by-step process `contact` takes you through

Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation question, contact the TAC. This group stands ready to solve such problems.

The `contact` Utility

The TAC recommends using the `contact` utility to report a hardware, software, or documentation problem. The `contact` utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke `contact`, it prompts you for information about the problem. When you finish your report, `contact` electronically mails it to the TAC. The TAC notifies you within 48 hours that your report has been received.

To use `contact` requires:

- UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- Full path name of the program or utility in question
- Version number of the program or utility in question

UUCP Connection

Before using `contact`, ask your system administrator if your site has a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX-based system to another. The `uucp` (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

Finding the Program Path Name

To determine the full path name of the program or utility in question, use the `which` command. Figure A-1 illustrates use of the `which` command to find the full path name of the loader (`ld`) utility.

Figure A-1
Using the `which` command

```
> which ld
/bin/ld
>
```

In this example, the full path name of the loader is `/bin/ld`.

If you use the C shell (`csh`), you can also use the `whence` command to find the program path name. The `whence` command works like `which`, but faster.

For more information on the `which` command, refer to the which(1) man page. You can also use the `info` online information system by entering **`info which`** at the system prompt.

Finding the Program Version Number

To determine the version number of the program or utility in question, use the `vers` command. Figure A-7 illustrates use of the `vers` command to find the version number of the loader (`ld`) utility. Enter **`vers`**, then the path name of the program or utility.

Figure A-2
Using the `vers` command

```
> vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader version number is 7.0.

For more information on the `vers` command, refer to the vers(1) man page. You can also use the `info` online information system by entering **`info vers`** at the system prompt.

Using contact

The contact utility prompts for the following information:

- Your name, title, phone number, and corporate name
- Name and version of the product
- One-line summary of the problem
- Detailed description of the problem
- Priority of the problem
- Instructions on how to reproduce the problem
- Comments about the problem
- Comments about the documentation relating to the problem
- Files to include in the contact report

Following is a step-by-step discussion of these prompts.

Step 1a

To invoke the contact utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software, or documentation question. Figure A-3 illustrates use of the contact command and the resulting system response.

Figure A-3
Beginning a contact session

```
> contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

Step 1b

If there is a .contact file in your home directory, contact skips the first prompt. (Refer to "Using a .contact File" on page 10 for more information.) Figure A-4 illustrates the contact command and the system response when you have a .contact file in your home directory.

Figure A-4
Beginning a contact session
with a .contact file

```
> contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

Step 2

The `contact` utility prompts for the version number of the product. If you do not know the version number, press **CTRL-Z** to suspend the session.

Use the `which` (or `whence` if you use `csh`) and `vers` commands to find the version number of the product. Use the `fg` command to return to the session, and enter the version number in the form X.X or X.X.X.X.

Step 3

The `contact` utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Please make this summary as descriptive as possible in one line.

Step 4

The `contact` utility prompts for a detailed description of the problem. Please make this description as complete as possible. Include source code and a stack backtrace when possible. (Refer to the adb(1) or csd(1) man page for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve the problem.

Step 5

The `contact` utility prompts for the priority of the problem. Figure A-5 illustrates this prompt and priority levels from which to choose. You must enter a priority number.

Figure A-5
Specifying priority of a problem

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

Step 6

The `contact` utility prompts for an explanation of how to reproduce the problem. Please include the command syntax and options you used and anything else you did to make the program run.

Step 7

The `contact` utility prompts for any other pertinent comments. Please include all relevant information.

Step 8

The `contact` utility prompts for suggestions regarding documentation supporting the product. Indicate whether the documentation could be revised to address the problem.

Step 9

The `contact` utility prompts for names of files necessary to reproduce the problem. Figure A-6 illustrates this prompt and sample user response.

Figure A-6
Including files in a `contact` report

```
Are there any files that should be included in this report (yes | no)?
> yes
Please enter the names of the files, one to a line (^D to terminate)
> test.f
> ~/subroutines/sub.f
>
```

Note

"Tilde-Escape Sequences" on page 11 are not recognized in responses to this prompt. In `contact`, a tilde in this section indicates your home directory. This convention is based on use of the tilde for expanding file names in `csh`.

If files specified are small text files, they are automatically included in the `contact` report. If the files are too large to be included in this report, `contact` gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the `tar` command (refer to the tar(1) man page for further information) or enter each file name in the directory on a single line in the `contact` report.

Step 10

The `contact` utility prompts you to review, edit, submit, or abort the report. Figure A-7 illustrates this prompt.

Figure A-7
Prompt to review, edit, submit, or abort report

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review	review the text of the `contact` report. You are then prompted again to select an option.
Edit	edit the text of the `contact` report. If you choose to edit the report, `contact` opens your default text editor.
Submit	sends the report to the CONVEX TAC. The TAC notifies you within 48 hours that your report has been received. Choosing this option exits the `contact` utility and returns you to the shell.
Abort	saves the text of the report in a file named ~/dead.report. Choosing this option exits `contact` and returns you to the shell.

Tips for Using `contact`

The `contact` utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to:

- Use a .contact file.
- Abort a `contact` session.
- Resubmit an aborted report.
- Suspend a `contact` session.
- Move within `contact` from one prompt to another.
- Use tilde-escape sequences in the `contact` utility.

Using a .contact File

When you invoke `contact`, it first prompts for your name, title, phone number, and company name. You can, however, create a .contact file to skip this first prompt.

Follow these steps to create a .contact file.

1. Create a .contact file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke `contact`, it automatically includes the .contact file as input for the first prompt and proceeds to the next prompt.

Aborting the Report

To abort a `contact` report, either press the interrupt key (usually **CTRL-C**) or choose the `abort` option when prompted by the `contact` utility. Using **CTRL-C** to abort does not save the contents of the report. Using the `abort` option saves the contents of the report in a file named ~/dead.report.

Submitting the dead.report File

After you abort a `contact` session, the `contact` utility saves the report in a file named ~/dead.report. Using the `contact` command with the `-r` option automatically merges the contents of the ~/dead.report file into the new `contact` session. Enter

```
contact -r
```

and `contact` finds the ~/dead.report file and merges it into the `contact` report. You can then edit the report. When you end the editing session, `contact` resumes at the final prompt, which asks you to review, edit, submit, or abort the report.

Suspending a Report

Sometimes it is necessary to stop in the middle of a `contact` report and return to the shell (for instance, to suspend the `contact` session to find the program path name or version number). To suspend the `contact` session, press **CTRL-Z**.

To return to the `contact` session, type **`fg`**. Using **CTRL-Z** and the `fg` (foreground) command, you can switch between the `contact` utility and the shell. You cannot, however, use **CTRL-Z** and `fg` to switch back and forth in the Bourne shell (`sh`).

Ending a Response

The `contact` utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press **RETURN**. Other prompts require more than a one--line response; to move to the next prompt, press **CTRL-D**.

Tilde-Escape Sequences

The `contact` utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. You can use the following tilde sequences within `contact`:

~e	start the text editor (defined in the EDITOR environment variable)
~h	display a list of available tilde-escape sequences
~p	print the `contact` report to the terminal screen
~r *filename*	read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response.
~~	insert a single tilde as the first character in the line

Index

T

U

V

X

Y