CONVEX Network File System
User's Guide

Document No. 710-001530-203

Second Edition, Rev. 2
November 1988

CONVEX Computer Corporation
Richardson, Texas

CONVEX Network File System
User's Guide
Order No. DSW-112
Second Edition, Rev. 2

Printed in the United States of America

Revision Information for
CONVEX Network File System
User's Guide

Edition	Document No.	Description
Second Rev. 2	710-001530-203	Released with CONVEX UNIX V7.0, November 1988. Includes the following changes: Chapter 3 updated sample *yp* domain maps updated sample *ypwhich* domain maps
Second Rev. 1	710-001530-202	Released with CONVEX UNIX V6.2, April 1988.
2.0	710-001530-201	Released with CONVEX UNIX V6.1, October 1987.
1.0	710-000630-000	Initial release with CONVEX UNIX V6.0, April 1987.

Table of Contents

Appendices

List of Tables

List of Figures

Preface

Intended Audience, Prerequisites

This document contains instructions for using the CONVEX Network File System (*nfs*), and user interfaces to both the Yellow Pages (*yp*) and the Remote Procedure Call (*rpc*) software. The Network File System is a means for sharing file systems among networked computers. With it, you can access remote files and directories easily, and you can use *nfs* as the basis for network-wide file sharing. Use of *nfs* is described in Chapter 2.

The Yellow Pages is a distributed data base system, useful for automating the administration of system files like */etc/networks*, */etc/groups*, and */etc/passwd*. Although the *yp* software is designed primarily for system managers and system programmers, you may find the associated utilities *ypcat*(1), *ypmatch*(1), *yppasswd*(1), *ypwhich*(1), and *ypclnt*(3N) helpful. The use of these utilities is described in Chapter 3.

The emphasis of this book is tutorial: if you have previous exposure to *nfs*, *yp*, or *rpc*, you may wish to use the *CONVEX Network File System Reference Set* as a supplement. The *Reference Set* contains protocol specifications for *nfs*, *yp*, *rpc*, *xdr*, and a guide to programming with *rpc*. If you are a system manager, you will definitely want to supplement this volume with the *CONVEX Network File System System Manager's Guide*, which explains how to install and troubleshoot *nfs* and *yp*.

Specific prerequisites to the use of this manual include familiarity with the C programming language and the CONVEX UNIX operating system.

Document Structure

This document is organized into three chapters, as follows:

- Chapter 1 introduces *nfs*, *yp*, *rpc*, and *xdr*, and the *virtual file system*, on which they are built.
- Chapter 2 describes how to use *nfs*.
- Chapter 3 describes how to use the *ypcat*(1), *ypmatch*(1), *yppasswd*(1), *ypwhich*(1), and *ypclnt*(3N) utilities.

All chapters contain numerous examples.

Notational and Typographical Conventions Used

The following conventions are used in this document:

- Mnemonics enclosed in "less than" and "greater than" signs designate ASCII nonprintable characters. For example, **<CR>** stands for "carriage return."
- Within command sequences set off from regular text, **boldface** type indicates literals. Words appearing in **boldface** must be typed just as they appear. *Italics* within command sequences indicate generic commands or filenames. Substitute actual

commands or filenames for the *italicized* words. For example, the command sequence:

ld [*switches*] [*object files*] [*libraries*]

instructs you to type the command *ld*, followed by your choice of switches, object files, and libraries.

Italics within text indicate commands, filenames, or programs.

- Brackets [ ] designate optional entries.
- A horizontal ellipsis ... shows repetition of the preceding item(s).
- A vertical ellipsis shows continuation of a sequence where not all of the statements in an example are shown.
- Commands, utilities, and files that are documented in the *CONVEX UNIX Programmer's Manual* are italicized; occurrences that include a number enclosed in parentheses refer to the appropriate section of the manual (for example, *vmstat*(1) means that the command *vmstat* is located in Section 1 of the *Programmer's Manual*).
- The | symbol is used to denote command sequences in which you must pick no more than one alternative from a list of command options. In the following command sequence, for example:

  `(fp)>` **s[et] s[pu-selftest]** = [*d*[*isable* |*e*[*nable*]

  you must choose either *d*[*isable*] or *e*[*nable*], but you cannot choose both.
- The pound sign (#) signifies the superuser prompt. The percent symbol (%) signifies the standard C shell user prompt.

Associated Documents

The following documents provide information that you will find useful as you learn about the Network File System and its related utilities.

- *CONVEX System Manager's Guide.* This manual provides all of the information needed to manage and maintain a previously installed and configured CONVEX system. Of particular relevance to successful operation of *nfs* are the chapters on system start-up and shutdown, and network (LAN) management.
- *CONVEX UNIX Programmer's Manual.* This multi-volume manual is the definitive reference for the CONVEX UNIX operating system. The *Programmer's Manual* contains all of the manual pages referenced in this book.
- *CONVEX Network File System System Manager's Guide.* This manual describes how to install and maintain *nfs* and *yp*.
- *CONVEX Network File System Reference Set.* This volume contains six reference manuals describing *rpc* programming and the *nfs*, *rpc*, and *xdr* protocols.

1

Introduction

This book describes the user interfaces to a set of optionally priced networking products. These products, included with the CONVEX Network File System product offering, include the following:

- Network File System (*nfs*), related utilities, and daemons. The Network File System enables you to share file systems across a network and to access these files and directories easily and transparently.
- Remote Procedure Call (*rpc*) software, related utilities, and daemons. The Remote Procedure Call software enables you to make procedure calls across a network.
- External Data Representation (*xdr*) software, related utilities, and daemons. The External Data Representation software provides a mechanism for translating data types and structures between machines with dissimilar architectures.
- Yellow Pages (*yp*) software, related utilities, and daemons. The Yellow Pages software enables you to create, modify, and maintain distributed databases.

These products are linked into an integrated networking system built around a *virtual file system* (*vfs*) architecture that enables the operating system to communicate with multiple file systems. In effect, the *vfs* architecture enables the operating system to communicate with file systems in much the same way that it communicates with device drivers. These file systems can be either local or remote (networked) and may be either UNIX or non-UNIX file systems. *nfs* and its related products (*rpc*, *xdr*) enable you to access these remote file systems either directly or through the use of system calls. The *yellow pages* database helps make the system work by keeping track of system support files stored on the network. *yp* also functions as a lookup service, providing you with easy access to password information and host addresses for the entire network.

Note that this book focuses on the *use* of these products. Other books (*CONVEX Network File System System Manager's Guide, CONVEX Network File System Reference Set*) deal with administrative issues, internals, and expert user scenarios. This book describes, with examples:

- How to use *nfs* (Chapter 2)
- How to use the various *yp* user interfaces: *ypcat*(1), *ypmatch*(1), *yppasswd*(1), *ypwhich*(1), and *ypclnt*(3N) (Chapter 3)

You may wish to refer to the protocol specifications for *nfs*, *xdr*, *rpc*, and *yp* as you work through this manual. These specifications are included in the *CONVEX Network File System Reference Set.*

Architectural Overview

CONVEX networking products are linked into a hierarchical networking system that consists of three primary components:

- The CONVEX UNIX operating system release, based on the Berkeley UNIX 4.2 operating system

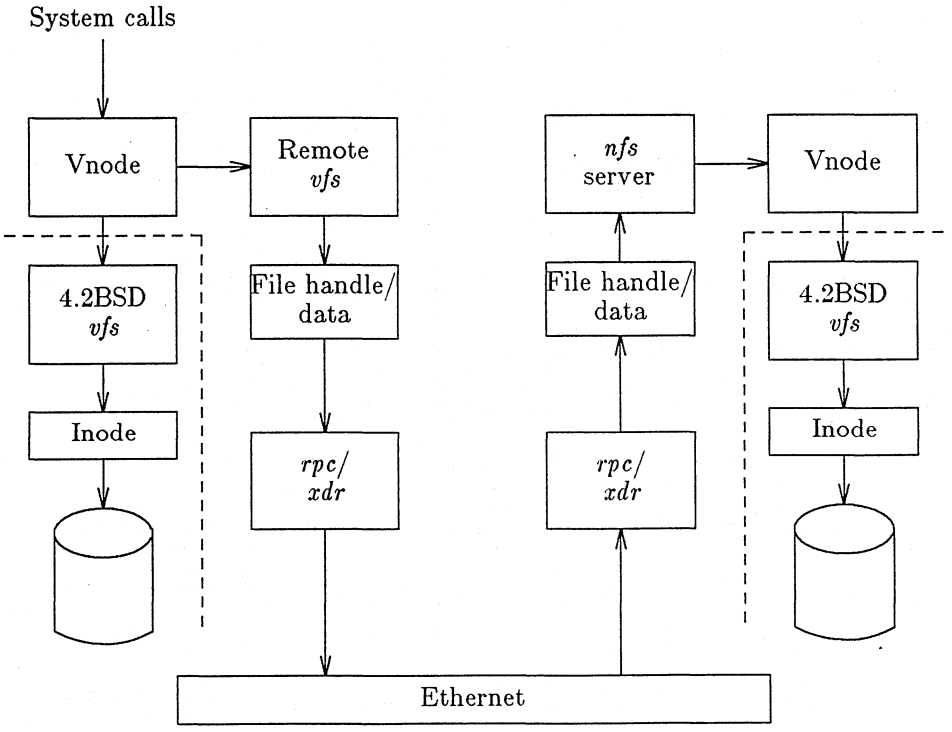
- A generic file system interface based on the *vnode* and *vfs* data structures
- The networking interface, which enables you to use *nfs*, *rpc*, *xdr* and the other networking products

CONVEX UNIX is documented completely in the current documentation set. For differences from earlier releases, see *Release Notices*. For system management information, see the *CONVEX System Manager's Guide*. For information on new programs and utilities, see the *Programmer's Manual*, Parts I and II.

The generic file system (called *vfs*) enables many file systems to be implemented within the CONVEX UNIX operating system *kernel*. The practical effect of this architecture is to allow a network of heterogeneous operating systems to communicate and share data. *vfs* accomplishes this through the use of *vnodes*, or virtual inodes, that provide a virtual index to files across the network. Vnodes work much as *inodes* do, but rather than indexing individual files, they index virtual file systems, either remotely or locally. Remote *vfs* calls connect you to file systems on remote systems; the local *vfs* connects you to file system data on the local device.

The virtual file system interface has been constructed as follows. The file system is split above the kernel *inode* layer. (See Figure 1-1.) Above the split, file system independent inodes (called *vnodes*) contain data fields needed to identify and reference files. Vnodes contain data pointers that point to data located either remotely or locally. Below the split, the file system works just as it did before.

Figure 1–1: Virtual File System Architecture



Note that *nfs* is built on top of *rpc/xdr*. *rpc* and *xdr* provide a low-level network interface used by *nfs*. *rpc* and *xdr* use *file handles* to implement the interface. File handles are data structures that contain the information needed for remote file systems to map file names to inodes.) In practice, this arrangement works much like the arrangement between networking products like

telnet or *ftp* and network protocols like TCP/IP. The higher-level programs (*telnet, ftp, nfs, yp*) retrieve and organize user input, which is passed to the lower level programs (TCP/IP, *rpc/xdr*) for transfer across the network. *nfs*, of course, offers much more functionality to the user than either *ftp* or *telnet*.

In operation, the virtual file system works as follows. If the vnode layer receives a call requesting local data, the call is passed to the local file system, which returns the data to the user. (Local file systems continue to access data via disk drivers, as they did before *vfs*.) On the other hand, if the vnode layer receives a call requesting remote data, the call is forwarded across the Ethernet to the remote host. There, the remote host accesses data through a disk driver and returns it across the Ethernet. *rpc* and *xdr* are used both on the remote call and the return to make the remote call and to translate data to a machine-independent format, respectively.

At the networking interface level, *nfs*, *rpc*, *xdr*, and *yp* become useful to the programmer working on individual problems. *nfs* enables the system manager to mount, unmount, and access remote file systems. *rpc* and *xdr* allow programmers to make remote procedure calls across the network. (Typical uses for these products include determining the number of users on a remote system, using the *rpc* batching mechanism to send batch jobs to a computational server, etc.) And the yellow pages, as mentioned earlier, function as a network-wide lookup service.

2

Using *nfs*

Overview

nfs is a CONVEX adaptation of the SUN Network File System (*nfs*) software. More specifically, *nfs* is a kernel-level implementation of SUN *nfs* client and server routines. *nfs* provides remote access to CONVEX files from any other machine supporting *nfs*.

nfs provides the benefits of other remote file access products (*rcp* or *ftp*) with one important advantage: file systems exported using *nfs* are not copied between systems. Instead, these file systems are networked across both machines. The result is a true shared file system that enables users to transparently access files and directories shared with a CONVEX server.

The following example illustrates the benefits of *nfs*. Suppose you are working on a workstation and you decide to make some modifications to a file containing documentation being prepared by your project group. Unfortunately, the file is located on the CONVEX supercomputer used by your group. If you have not installed *nfs* on the CONVEX supercomputer, your choices at this point include:

- Performing some type of remote login to the CONVEX supercomputer, or
- Performing a remote copy operation (using *ftp* or *rcp*)

Obviously, each of these alternatives has disadvantages. If you choose to log in remotely, you are faced with saving a file, logging in, modifying the file, and logging out before you can resume work. If you choose the latter alternative, you have to copy the file, save it, change it, and then copy it *back*. Worse, you're only capable of changing one copy of the file at a time. If several members of your group are working on the file, it's nearly impossible to get anything done unless the file is centrally located—giving each person their own copy of the file is out of the question.

If, on the other hand, you have installed *nfs*, you simply edit the file as if it was located on your workstation's disk. The file is, of course, located on the server, but since *nfs* file systems are networked, all of the workstations on a network can share a common version of any particular file. Furthermore, sharing is accomplished with a high degree of transparency—files *appear* to be located locally, but changes to the "local" file change the server's file system, which is shared by all clients.

nfs consists of three daemons—*portmap*(8C), *nfsd*(8), and *biod*(8). *portmap* maps *rpc* program numbers and program versions to Defense Advanced Research Project Agency (DARPA) port numbers. *nfsd* is the file server proper, which watches for and services requests from remote systems. *biod* handles I/O on *nfs* clients. These three daemons are included on the *nfs* distribution tape.

Using *nfs*

Before you can use a remote directory, you must mount it using the *mount*(8) command. *mount* causes a server directory to be "attached" to an existing directory tree on the client system. The *umount*(8) command, of course, reverses this procedure and unmounts a previously mounted directory. The basic use of both of these utilities is described here.

Several caveats apply to mounting and unmounting directories under *nfs*. First, note that you *must* mount only *directories*: if you attempt to mount files, the results are guaranteed to be unpredicatable. Second, remember that you must use absolute pathnames; *mount*(8) does not accept attempts at relative addressing on either client or server. Third, note that *umount* is always unsuccessful if the mounted file system is currently in use. This means that the directory to be unmounted cannot be the working directory of any active process. Finally, note that you cannot use *flock*(2) to place an advisory lock on a networked file.

NOTE

Directories should be mounted and unmounted by a trained system manager. The process of determining which directories should be networked requires careful planning by individuals with an in-depth understanding of the CONVEX UNIX system. For these reasons, you should **not** mount or unmount a directory, even if you have superuser privileges, without consulting with your system manager first.

Using the *mount* Command

mount uses a standard syntax:

/etc/mount *mount_option server:pathname directory*

where:

mount_option refers to the type of mount to be accomplished—soft, hard, or interruptible hard (the various mount options are discussed subsequently)

server is the name of the file server you wish to use

pathname is the name of the server directory you wish to mount (this directory must exist when the *mount* command is issued)

directory is the name of a client directory (this directory must also exist when the *mount* command is issued)

You should keep in mind a few facts about mount operations. First, server directories are "attached" to the client file tree at the *mount point*, which is the client pathname you've specified. (That is, *pathname* is attached to the client file tree at *directory*.) Second, you can mount a directory "soft," "hard," or "interruptible hard." When you specify a "soft" mount, *mount* returns an error if the server doesn't respond within a specified time-out period. "Hard" mounts continue to retry mount requests until the server responds. "Interruptible hard" mounts function exactly like hard mounts, but can be interrupted from the keyboard. (For more information about mount options, see the *CONVEX Network File System System Manager's Guide*.)

Finally, the top-level directory attached is referred to on the client machine with the name of the directory used as a mount point. Here's an example. Suppose you use the following command sequence to mount a server directory:

client# **/etc/mount convex:/mnt/ed/horse_of_course /mnt/ed/glory_days**

to mount user Ed's directory, *horse_of_course*, on the client. Obviously, *horse_of_course* and its subdirectories and files are mounted just "below" *glory_days*. Less obviously, the directory name *horse_of_course* does not migrate along with its directory. On the client, the directory is referred to as *glory_days*. Figure 2-1 shows this situation graphically.

Figure 2–1: Sample Network File System

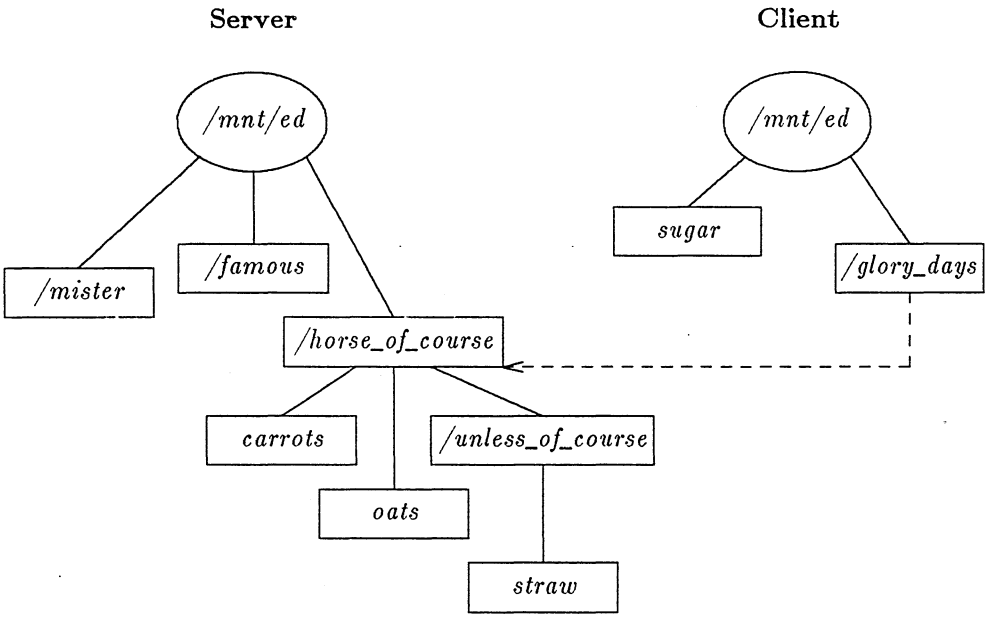


Figure 2-1 shows the exported server file system and its companion file system. Since the server file system has been exported, files like *carrots*, *straw*, or *oats* are shared between client(s) and server, and changes to them can be made from either machine. Note, though, that the pathnames for the data nodes vary from machine to machine. To read *oats* on the server, for example, you would issue a command similar to:

% **more /mnt/ed/horse_of_course/oats**

On the client, however, the corresponding operation is performed as follows:

client% **more /mnt/ed/glory_days/oats**

You do *not* use the directory name *horse_of_course*. This name has no meaning on the client—the last server-side node referenced in the *mount* command is always renamed.

Using the *umount* Command

You can use *umount* in several ways. First, to unmount all the directories remotely mounted from a particular server, log in as superuser and use the following syntax:

client# **/etc/umount -h** *server_name*

where *server_name* is the name of the file server in question. For example, to unmount all the directories mounted from a system named "convex," enter:

client# **/etc/umount -h convex**

To unmount a particular directory, vary the syntax as follows:

client# **/etc/umount** *dir_name*

where *dir_name* is the name of the directory used as a mount point for the remotely mounted directory. If you are not sure what this name is, a check using *df*(1) gives you output similar to the following:

```
Filesystem          kbytes  used    avail   capacity Mounted on
/dev/da0a           20415   11944   6429    65%      /
/dev/st9            249487  204397  20141   91%      /os
/dev/st8            249487  213336  11202   95%      /scratch
/dev/st7            249487  196836  27702   88%      /lp
/dev/st6            163423  136242  10838   93%      /dmaster
/dev/st4            251535  185332  41049   82%      /diag
/dev/st3            41614   12336   25116   33%      /tmp
/dev/st2            207819  68984   118053  37%      /nfs
convex:/usr         86607   47948   29998   62%      /usr
convex1:/mnt/ed/horse_of_course
                    108645  87083   10697   89%      /mnt/ed/glory_days
```

If you want to unmount the directory *horse_of_course*, for example, type:

client# **/etc/umount /mnt/ed/glory_days**

Using Advisory Record Locking

CONVEX supports a record-locking system that enables you to create advisory locks that are compatible with System V implementations. This system enables you to lock records in a network environment, making distributed databases using *nfs* more feasible and useful. Utilities used to support record locking include *fcntl*(2) (executes file and record-locking requests), *lockf*(3) (a user-friendly front end to *fcntl*), *lockd*(8C) (the network lock daemon), and *statd*(8C) (used as a status monitor). The use of *lockf* is described here. For information on installing or administering the lock manager, see the *CONVEX Network File System System Manager's Guide.*

How Record Locking Works

Record-locking mechanisms enable users to control access to particular file regions, or *records.* Because UNIX has no concept of records, records are defined as arbitrary sequences of bytes located at the current file pointer. (Note that record-locking schemes differ from *file-locking*

programs, which prevent processes from reading or writing entire files.) *Advisory* record locking enables cooperating processes to implement record locking, but the kernel does not in any way force other processes to respect the advisory locks. This means that errant processes may access previously locked records without using advisory locks, possibly resulting in database inconsistencies. Note, however, that in practice, application programs running on multiple *nfs* clients can cooperate to lock and update records of a master database effectively.

CONVEX supports two types of advisory locks: *shared* and *exclusive* locks. Shared locks enable cooperating processes to read locked records. Exclusive locks are placed when processes either write, or read and write, records. More than one process may hold a shared lock at any given time; but multiple exclusive, or shared and exclusive, locks may not exist simultaneously on the same record.

How *lockf* Works

The *lockf*(3) library routine is a front end, or user interface, to the *fcntl*(2) system call, which performs file and record locking. *lockf* is used to place advisory locks on *records*. In this respect, it as an improvement over *flock*, which provides a file locking mechanism only. *lockf* works by enabling you to specify the number of contiguous bytes within a particular file to be locked or unlocked. This data is passed in the *size* argument and may be positive (to indicate bytes extending forward from the current file offset) or negative (to indicate bytes preceding but not including the current offset). Logically, of course, these regions of contiguous bytes represent records. (Note that if you expand these regions of contiguous bytes far enough, you have, in effect, implemented file locking; *lockf* can be used for file locking using this method.)

lockf enables you to lock and unlock file regions; test a region for other locks; and to test and then lock a region. Test-and-lock operations that fail with a *-1* return the *No more processes* (EAGAIN) diagnostic if the section is already locked by another process. Lock operations sleep until locks are granted or a signal is caught.

Note that *lockf* does not offer all of the functionality of *fcntl*. In particular, *lockf* restricts you to:

- Use of exclusive locks (*lockf* does *F_WRLCK fcntl()* requests only)
- Starting at the current position in the file

Note further that *lockf()* returns the *EDEADLK* diagnostic, not the *ENOLCK* diagnostic returned by *fcntl*. (*flock* returns *EDEADLK* to ensure compatibility with */usr/group* standards.) Typically, complex tasks like specifying shared record locks or using a file offset different from the current position are best handled with *fcntl*.

Note that your application may be sent a *SIGLOST* signal if a file lock on an *nfs* file cannot be reclaimed after a server crash. This signal is not standard to System V—it is an extension developed when record locking was extended into the network environment using *nfs*. If you want to use *SIGLOST* in your application, you can use the *#ifdef* C preprocessor feature as shown in Figure 2-2. Using *#ifdef* enables you to use *SIGLOST* while maintaining System V source code compatibility.

Figure 2–2: Adding *SIGLOST* Signal to Application Programs

```
#include <signal.h>
#ifdef SIGLOST
int lostlock();
#endif
#ifdef SIGLOST
signal(SIGLOST, lostlock)
#endif
#ifdef SIGLOST
lostlock()
{
/*
 * Code here to handle lost record lock; would perhaps
 * try to regain its lock again or exit with a message
 */
}
#endif SIGLOST
```

rex: Remote Execution Utility

rex(1C) and its accompanying daemon, *rexd*(8C), enable you to execute commands remotely, providing you with the opportunity to off-load *nroff*, *make*, and other jobs onto lightly loaded machines. Unlike *rsh* and *rlogin*, *rex* neither starts a shell on the remote host nor performs remote logins. Instead, *rex* accomplishes the remote execution task by mounting local file partitions on remote machines via *nfs*. In an open environment, *rex* can also be used to improve load balancing between machines. *rex* does not require installation and uses a simple syntactical structure. Subsequent sections describe the use and administration of *rex* and *rexd*.

Like *rlogin* and *rsh*, *rex* operates quickly, without discernible start-up time. In fact, because *rex* doesn't perform remote logins or start up remote shells, it is faster than either *rsh* or *rlogin*. Note, however, that while *rex*'s quick start-up time is certainly an advantage, *rex* users are limited to relatively simple command sequences. Because *rex* does not start a remote shell, it cannot interpret the complex command sequences interpreted by *rsh*.

rex and *rexd* are included by default on standard *nfs* release distribution tapes. Because no installation is required, you should be able to use these utilities immediately. If you have trouble locating or using *rex* or *rexd*, contact the CONVEX Technical Assistance Center.

Using *rex*

The standard syntax for using *rex* is:

% rex -i -n -d *host command arguments*

where:

-i Is used to invoke *interactive* mode (described subsequently).

-n Is used to specify *no input*. This option is used to "close" standard input so that jobs can be run in the background without generating errors. (Ordinarily, remote standard input is passed from *rex*'s standard input.)

-d Is used to invoke *debugging* mode. In debugging mode, *rex* prints diagnostic messages as work is being done.

host Is the name of the remote host jobs are to be run on.

command Is the command to be run.

arguments Are standard command-line arguments (for example, the *-la* arguments commonly used with *ls*).

rex is commonly used to off-load high-overhead jobs (*nroff* and *make* jobs are prime examples). If, for example, you want to run an *nroff* job on a processor named *lotsacycles*, use *rex* as follows:

```
% rex lotsacycles nroff *.s
```

Although *rex* is convenient, you might have tried to use *rsh* to perform the same task. Unfortunately, *rsh* would not have worked because the two utilities work differently—*rsh* executes commands in your *home* directory on the remote host, while *rex* executes commands after mounting your current *local* working directory on the remote host. In the example above, *rsh* would have looked for *.s* files in your home directory on the remote machine, and never would have found the local files you wanted to process. In fact, unless your remote home directory contained some *.s* files, *rsh* probably would have failed to do anything except return an error message. Table 2-1 illustrates the differences between *rex* and *rsh*:

Table 2-1: Differences Between *rex* and *rsh*

rex	rsh
Executes command with arguments on host after mounting current local working directory and file system (if not already mounted).	Executes command with arguments in home directory on host.
Can't execute shell built-ins (e.g., *alias*).	Can execute shell built-ins because default shell is started on host.
Inherits current directory and ENVIRONMENT variables.	Does not inherit current directory or ENVIRONMENT variables.
Runs interactive commands.	Cannot run interactive commands.
Propagates terminal mode and window size to remote host via interactive commands.	Cannot perform these functions.

The important thing to remember is that while *rsh* has no concept of networked file systems, *rex* always uses them. (If a local file system is not mounted on the remote host, *rex* always tries to mount it.) *rex* always works from *local* files, *on* a remote machine. *rsh* always works from *remote* files, *on* a remote machine.

Because *rex* works from remotely mounted versions of the local file system, you should use relative pathnames that exist within the current file system. Note that *rex* mounts entire file systems, whether or not you are at a mount point. For example, the following command sequence:

```
% pwd
```

```
% /mnt/moe/src
% rex make all
```

mounts */mnt*, not */mnt/moe/src*. Once the file system is mounted, *rex* changes directories to the current working directory.

You can also use *rex* to run interactive jobs (like text editors) remotely. Command sequences similar to the following enable you to edit *local* files on a remote host interactively:

```
% rex -i convex vi *
```

To edit *remote* files, use a command sequence similar to:

```
% rex -i convex vi /etc/hosts
```

This command sequence enables you to interactively edit *convex*'s */etc/hosts* file. Using this approach, you can more easily complete tedious tasks like editing messages of the day around the network.

Examples of Advanced *rex* Usage

Figure 2-3 shows how *rex* can be used in a shell script to simplify day-to-day programming tasks. In this example, *rex* is used to run a job on the most lightly loaded machine available from the set including *fred*, *barney*, *pebbles*, and *bambam*. To use the script, add it to your */bin* directory as *qrex*, and invoke it via:

```
% qrex job_name
```

Figure 2–3: *rex* Application Example

```
#!/bin/sh
#
# qrex -- run a rex job on the lowest loaded machine that is listed as
#                                          a good candidate.
#
# The HOSTS variable contains an egrep format string of hosts that run the
# rex daemon and can compile sources to produce Convex binaries.
#
HOSTS="^fred |^barney |^wilma |^pebbles |^bambam "
REXHOST=`ruptime -l | egrep "$HOSTS" | grep " up " | tail -1 | awk '{print $1}'`

if [ -z "$REXHOST" ]; then
  echo $0: No host available 1>&2
  exit 1
fi

rex $REXHOST "$@"
```

rex can also be used with symbolic links to shorten command sequences that you use frequently. The following command sequence, for example:

```
% ln -s /usr/bin/rex ~/bin/convex
```

establishes a symbolic link between *rex* and a command name (*convex*) in your local */bin* directory. Having created the link, you can shorten command sequences referencing the host, *convex*. For example, once you've created the symbolic link, you can type:

```
% convex make
```

instead of:

```
% rex convex make
```

If you decide to use symbolic links, you'll obviously need symbolic links for every host you intend to use.

Disk Quotas and *nfs*

The CONVEX UNIX operating system provides a disk quota system to help system managers control the amount of disk space consumed by their users. This system warns you when you exceed the amount of disk space, or inodes, or both, allotted to you.

Typically, the quota system is set up with both "soft" and "hard" limits. (Note that these limits have no relation to the "soft" and "hard" options used with the *mount* command. To enable the quota, however, you must mount partitions with the "quota" option enabled. See *mount*(8) for more information.)

Soft limits are set in numbers of 1K bytes. If you exceed the soft limit, a warning message is sent to you, and a time limit is set. Messages continue to be sent until you reduce your resource usage below the soft limit. If you ignore the warning messages and eventually exceed the time limit as well, the soft limit is treated as if a hard limit had been reached, and further resources are denied to you.

Hard limits cannot be exceeded. When you reach a hard limit, further requests for space, or attempts to create a file, fail with an EDQOUT error. The first time this occurs, a message is written to your terminal. Only one message is written, and resources are denied until you reduce your disk space consumption below the hard limit.

Disk quotas enforced via *nfs* on remote file systems differ in two important ways from the standard disk quota system described above. First, no warnings are printed when you go over soft limits on remote file systems. To check your disk usage, you should use *quota*(1). Second, if you exceed a hard limit, you receive *nfs* return code errors, not system error messages. (If you use *write*(2), *fsync*(2), and *close*(2) system calls, you receive EDQUOT error return code. Make sure to check these return codes when you receive them.)

Information on setting up and administering disk quotas is included in the *CONVEX System Manager's Guide.*

Troubleshooting

Generally speaking, the problems you are likely to encounter in your use of *nfs* are minor. Sometimes, as with a hardware failure, or a problem with the Ethernet, you may run into some difficulty. With the product itself, however, correcting errors is generally a matter of modifying a file or two, or retyping a command line.

The next section describes system-level problems that are likely to surface when you attempt to mount directories. User-level problems are described next, and a table summarizing error conditions and symptoms completes the discussion. If you need more information than you can find here, refer to the *CONVEX Network File System System Manager's Guide.*

System-Level Problems

If any problems with the installation or hardware exist, you are likely to spot them when you try to mount a server file partition. The types of problems that can occur include:

- Forgetting to log in as superuser (remember that you must log in as root to mount and unmount file systems)
- Attempting to mount the partition in the current working directory
- Attempting to mount a partition not listed in the server */etc/exports* file
- Attempting to mount a partition when the server is not listed in the client */etc/hosts* file
- Attempting to mount a directory that has already been mounted
- Encountering a stale *nfs* file handle ("Stale" file handles are the structures remaining when a file is removed from a host without its handle being removed from a server.)
- Attempting to mount a directory when the Ethernet is down, when the server has crashed, or when the *nfs* daemons are down

You may forget to log in as superuser before you attempt to mount a partition. If you do, the system warns you as follows:

```
client% /etc/mount convex:/mnt/kirk/captains_log /mnt/stardate_2116
must be root to use mount
```

If you see this message, your best response is to log in as superuser and try again:

```
client% su root
client#/etc/mount convex:/mnt/kirk/captains_log  /mnt/stardate_2116
client#
```

A second common mistake is to try to mount a server partition in the current working directory. If you attempt this, the system responds as follows:

```
client% pwd
/mnt/docs/uhura
client% su root
client# /etc/mount convex:/mnt/uhura /mnt/docs/uhura
mount: convexe:/mnt/uhura on /mnt/docs/uhura: Mount device busy
mount: giving up on:
   /mnt/docs/uhura
```

The obvious solution here is to change working directories and try again:

```
client% cd ..
client% su root
client# /etc/mount convex:/mnt/uhura /mnt/docs/uhura
client#
```

You may have forgotten to include partition entries in the */etc/exports* file on the server.

(Remember: you must include an entry for each partition that you want to export.) In this case, you receive the following message:

```
client#/etc/mount convex:/usr/spool/mail /usr/spool/mail
mount: access denied for convex:/usr/spool/mail
mount: giving up on:
  /usr/spool/mail
client#
```

If you have forgotten to include the server name in the client */etc/hosts* file, the system reminds you as follows:

```
client#mount convex:/usr/spool/mail /usr/spool/mail
mount: convex not in hosts db
mount: giving up on
  /usr/spool/mail
```

If you encounter this problem, add the server name you wish to access to the */etc/hosts* file on the client and try again.

You may accidentally try to mount a directory that has already been mounted. If, for example, you try a second time to mount the directory */mnt/tracy/bandits/pruneface*, the following error message prints:

```
mount: convex:/mnt/tracy/bandits/pruneface already mounted
```

Luckily, this error requires no remedy beyond honest contrition.

You may receive the error message:

pathname: Stale NFS file handle

This message prints when an invalid file handle is passed to the server. Invalid file handles are sometimes passed when the file server daemon is restarted. The workaround is to unmount and remount the offending directory. For more information on file handles, valid and invalid, see the *CONVEX Network File System Reference Set.*

If you try to mount a directory while the Ethernet is down, the system seems to hang (i.e., blinking cursor, no keyboard echo, etc.). If this happens, check the Ethernet cable on the back of your terminal—it may be loose or disconnected. If the Ethernet cable seems to be connected, contact your system administrator.

Finally, if you attempt to mount a directory when the *nfs* daemons are down, you receive the following error message:

```
Port mapper failure - RPC: Timed out
mount: retrying
      pathname
```

To exit this process, type ^C (control-C).

User-Level Problems

Problems that you may encounter in your use of *nfs* include the following:

- Ethernet failure
- Attempt to access directories on machines without *nfs* installed
- Attempt to access directories on machines that are down
- Attempt to delete a directory used as a mount point

If you try to access a previously mounted partition while the Ethernet is down, the following message prints:

```
NFS server server_name not responding still trying
```

Your best response here is to check the Ethernet cable and try again.

If you try to access a directory on a machine without *nfs* installed, the system appears to hang for a few minutes; then the following error message prints:

```
mount: server: pathname server not responding:
Port mapper failure - RPC: Timed out
mount: retrying
     /mnt/fields/put
client#
```

Occasionally, users may try to delete a directory used as a mount point. If they do, the `Mount device busy` message prints. To delete the directory, first unmount the file system, then delete it from the server machine.

Troubleshooting Summary

Table 2-2 summarizes solutions to the common *nfs* problems discussed previously.

Table 2-2: Common Installation and User Errors

Error Message/System Behavior	Likely Cause/Solution
`access denied for` *system: filename*	Server */etc/exports* file incomplete. Modify file and try again.
`Mount device busy`	Attempted to mount/unmount a directory while a process is using it. Terminate all processes using directory in question. Message also prints when users try to delete a mounted directory (directory must be unmounted before deletion).
`mount:` *pathname* `already mounted`	Attempted to mount a directory previously mounted.
`NFS server` *server_name* `not responding, still trying`	Ethernet failure or downed server. Check Ethernet. Contact system administrator if necessary.
`must be root to use mount`	Attempted to mount partition without superuser login. Log in as superuser and try again.
`permission denied`	Attempted file access without permission or superuser access across network. Check file protection levels. If problem persists, check client and server */etc/passwd* and */etc/group* files for mismatched user or group IDs.
`Port mapper failure - RPC: Timed out`	Attempted to access machine without *nfs* installation or when server or server *portmap* daemon was down. Use *rpcinfo* to determine status of server's *portmap* daemon.
`Stale NFS file handle`	Invalid file handle was passed to server. Unmount and remount directory in question
system `not in servers db`	Server not listed in client */etc/hosts* file. Modify file and try again.
System seems hung; no response	Ethernet or server down. Check hardware and try again.

3

Using the Yellow Pages

Introduction

This chapter describes how to use four interface programs designed to help you access or modify data stored in yellow pages databases. Specifically, these programs—*ypcat*(1), *ypmatch*(1), *yppasswd*(1), and *ypwhich*(1)—enable you to access selected data from various databases, modify your yellow pages password file, and to determine which machine is acting as a database server. None of these programs demand more than a casual knowledge of the yellow pages. You will, however, need to know how the yellow pages work in general, what yellow page *maps* and *domains* are, and how maps and domains can be used. These topics are described in the next two sections of this chapter. (You can find more information in the *CONVEX Network File System System Manager's Guide.*)

The last program covered in this chapter, *ypclnt*(3N), is actually a library of functions that enables you to create your own interface to the yellow pages. Although *ypclnt* is more complicated to use than the other programs, individuals familiar with C and the CONVEX UNIX operating system should have no problems with it.

Introduction to the Yellow Pages

The yellow pages software package is a distributed network database system that enables you to look up information in a database from any network node. As with *nfs*, databases are shared across the network and are automatically updated after modification by users on the map's master server. The yellow pages software operates around a set of standard access procedures that hide the details of how and where data is stored. In practice, this means that you need to know very little about the configuration of any given network to access information from its databases.

The yellow pages software is typically used to distribute system administration files or other key-oriented data that is of interest across the network. Usually, many databases are distributed. You can use the utilities described in this chapter to figure out what information is stored in the various databases on your network, and to access that information.

The yellow pages software is organized around a system of *server* and *client* machines. *Servers* are machines that store, administer, and update the databases. *Clients* are machines that run *yp* processes to request data from databases on other machines.

Each yellow pages database can be stored and administered on multiple servers. This capability enables system managers to install many servers on a network, which increases reliability and performance. In practice, yellow pages servers can be serviced by any one of the servers on the network. Because the databases are shared among the servers, it doesn't matter which server process answers a client request; the answer is the same from any one of them.

Yellow pages databases are called *maps*. Maps are organized into *domains*. To use the yellow pages, you need to understand how these two concepts apply to your network. The concepts are explained in the next section.

Maps and Domains

Maps

As mentioned previously, *yp* databases are called *maps*. Maps are used primarily to simplify system administration across a network. In many cases, maps are built from files that formerly resided in */etc* directories. (As you recall, the */etc* directory typically houses password, group, host, and other files used for system administration.)

Each map contains a set of keys and associated values. For example, in a map called *hosts.byname*, all the host names within a network are the keys, and the Internet addresses of these host names are the values. Each *yp* map has a *mapname* used by programs to access it. You can use the several of the utilities described in this chapter to determine mapnames.

Maps become useful in administering networks in the following way. Suppose you are a system manager at a site running a network consisting of a dozen or so workstations linked to three CONVEX supercomputers, which are used as computational servers. Now suppose a new employee shows up and says she wants to "get on the system." Fine. But where do you add her password account? On one of the workstations? Sure, but which one? And what if she wants to access one of the CONVEX supercomputers? Typically, a good system manager tries to figure out *which* machines a user should access; he then adds user accounts to *each* of those systems. Unfortunately, he does this knowing full well that if the user changes her mind, or needs to change a password, he may have to undo his work on several machines. And, as the network grows, the problems are compounded.

Now, suppose you install *yp*. With *yp*, it becomes a simple matter to install a password map on the network database server, or perhaps on several servers. This map enables you to maintain password accounts for the entire user community in *one place*. You can also set up maps containing user *group* configurations, network configurations, and host names. Most sites running yellow pages create maps to serve these accounts, plus many others.

yp can, of course, be used to create *any* type of map. Although *yp* maps are used most often to distribute system administration information, they can also be used to distribute any type of information that is key-oriented and of interest to a network-wide community. For information on creating personal maps, see the *CONVEX UNIX Network File System System Manager's Guide* and the section on *ypclnt*(3N) at the end of this chapter.

Domains

In *yp* parlance, the subdirectory where a particular set of maps is stored is called a *domain*. (A single host can service many domains, just as it can contain many directories.) The general location for maps is /etc/yp/*domainname*.

You can check domain names via the command sequence:

```
% domainname
```

Suppose, for example, that you are logged onto a *yp* server named *convex*. If you type:

```
% domainname
```

and *convex* is printed, you know that you are in the *convex* domain. From there, you can look in */etc/yp/convex* for a listing of maps, or you can use *ypcat*(1) as described below. (You can, of course, also access databases stored in other domains.) Note that although many domains share a name with their host machine, domains may have any name. In the previous example, the

domain name might have been *engineering, comp_chem, passwd,* or *foobar.* In any case, though, the domain in question should be stored in a subdirectory of the same name.

Here's an example:

```
% ls /etc/yp
Makefile      makedbm*          revnetgroup*      ypnew         ypxfr.log
aliases.time  netgroup.time     rpc.time          yppoll*       ypxfr_1perday*
convexs/      networks.time     services.time     yppush*       ypxfr_1perhour*
ethers.time   passwd.time       stdhosts*         ypserv.log    ypxfr_2perday*
group.time    protocols.time    tinman/           ypset*
hosts.time    pwrestrict.time   ypinit*           ypxfr*
%
```

Note the *convexs* and *eclipse* subdirectories, which contain the maps for the *convexs* and *eclipse* domains, respectively. If you *cd* into the */eclipse* subdirectory, you find the following map listing:

```
% ls /etc/yp/eclipse
bootparams.dir      netgroup.byhost.dir      protocols.bynumber.dir
bootparams.pag      netgroup.byhost.pag      protocols.bynumber.pag
ethers.byaddr.dir   netgroup.byuser.dir      pwrestrict.byname.dir
ethers.byaddr.pag   netgroup.byuser.pag      pwrestrict.byname.pag
ethers.byname.dir   netgroup.dir             pwrestrict.byuid.dir
ethers.byname.pag   netgroup.pag             pwrestrict.byuid.pag
group.bygid.dir     networks.byaddr.dir      rpc.byname.dir
group.bygid.pag     networks.byaddr.pag      rpc.byname.pag
group.byname.dir    networks.byname.dir      rpc.bynumber.dir
group.byname.pag    networks.byname.pag      rpc.bynumber.pag
hosts.byaddr.dir    passwd.byname.dir        services.byname.dir
hosts.byaddr.pag    passwd.byname.pag        services.byname.pag
hosts.byname.dir    passwd.byuid.dir         services.dir
hosts.byname.pag    passwd.byuid.pag         services.pag
mail.aliases.dir    protocols.byname.dir     ypservers.dir
mail.aliases.pag    protocols.byname.pag     ypservers.pag
```

Yellow Pages User Programs

ypcat

ypcat enables you to print values from the yellow pages maps stored on your machine or on other machines across the network. *ypcat* is typically used with a map name as an argument. The command sequence:

```
% ypcat hosts.byaddr
```

for example, prints the contents of the map *hosts.byaddr,* which contains a listing of the names and addresses of network hosts. You can also invoke a map by using a "nickname." For example:

```
% ypcat hosts
```

has the same effect as the previous example; *hosts.byaddr* and *hosts* are just different ways of referring to the same map. You can list the nicknames for maps in your domain by using the *-x* argument with *ypcat*. For example:

```
% ypcat -x
Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byaddr"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
```

If you invoke *ypcat -x* on your machine, you get a good look at the maps that are available to you.

To display maps from different domains, use the *-d* switch. If, for example, you wanted to list *networks.byname* from the *eclipse* domain, you would use the following command sequence:

```
% ypcat -d eclipse networks

sun-ether     192.9.200     sunether ethernet localnet
loopback      127           loopback-net
sun-oldether  125           sunoldether
mktg-net      105
dragon-net    102
convex-net    100
ucb-ether     46            ucbether
arpanet       10            arpa
loopback-net  127
dragon-net    102
hyper-net     101           hyper
convex-net    100           convex
arpanet       10            # the arpanet
```

ypmatch

ypmatch, like *ypcat*, enables you to access data stored in databases not only on your machine but on other machines connected to the network. Unlike *ypcat*, *ypmatch* enables you to get at *selected* parts of the database by printing information that matches *keys* that you provide. Suppose, for example, you want to check the Internet address of the machine *eclipse*. You could log in to *eclipse* remotely, or you could use *ypcat* to print the contents of the *hosts* database on your machine. A simpler method is to use *ypmatch* to search the *hosts* database for values associated with the *eclipse* key. For example:

```
% ypmatch eclipse hosts
100.0.5.1     eclipse
```

The keys you supply *ypmatch* must exactly match the keys in the map. If you supply a key for which no values can be associated, you receive an error message:

```
% ypmatch loopback networks
ypmatch: Can't match loopback. Reason: no such key in map.
```

As with *ypcat*, you can use the *-x* option to display the map nickname table:

```
% ypmatch -x
Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byname"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
```

You can also use the *-d* flag to access data in a remote database, just as you did with *ypcat*:

```
% ypmatch -d eclipse docstaff aliases
 moe, curly, larry
```

When you specify the *-k* option, the key is printed (with a colon) before the value of the key. This option is useful primarily for improving the clarity of the output you receive when you specify more than one key. Here's an example:

```
% ypmatch -k moe curly larry passwd
moe: moe:VPF75JjbUCrvw:107:51:Moe,88/487,295,3415009:/doc/moe:/bin/csh
curly: curly:WTqWgH2mHeFRc:172:51:Curly,89/488,296,5307495:/doc/curly:/bin/csh
larry: larry:VL.fEiCaWwb/w:133:51:Larry,90/49,297,5273452:/doc/larry:/bin/csh
```

yppasswd

You can use *yppaswd* to change or install your yellow pages password. (As you remember from the previous discussion of maps, user passwords are usually one of the first things networked when *yp* is installed. The yellow pages password is much like a typical machine password, but it can be used to sign onto many machines across the network.) Your yellow pages password may be different from the one on your own machine.

Adding (or changing) a *yp* password is just like adding or changing your password on a CONVEX supercomputer. You type *yppasswd* at a command prompt to invoke the program; then you type your old password, followed by the new one. As with *passwd* on a CONVEX supercomputer, you must type the new password twice. Here's an example of a typical *yppasswd* terminal session (note that you may receive some messages not listed here, depending on how you use the command):

```
% yppasswd
  Old yp passwd: type_old_password_here
  New yp passwd: type_new_password_here
  Retype new passwd: type_new_password_here
```

Your new password must be at least six characters long. If you are not the password "owner," you must be a superuser to change a password. In either case, you must prove you know the old password. Note that the update protocol passes all the information to the server in one *rpc* call, without ever looking at it. Thus if you type your old password incorrectly, you are not notified until after you have entered your new password.

If you try to change your password on a machine that is not running the *yppasswd* daemon (*/usr/etc/rpc.yppasswdd*), you receive a message similar to the following:

```
% yppasswd
convexs is not running yppasswd daemon
```

If you receive this message, log in to the *passwd* master server and try again.

ypwhich

ypwhich provides a general query function that enables you to determine which maps are available to you, which server is supplying yellow pages services, and which machine is acting as master for a particular map.

In the simplest case, you can use *ypwhich* just as you did *ypcat* and *ypmatch* to display a list of maps and their nicknames. You do this by using the *-x* flag, as follows:

```
% ypwhich -x
Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byaddr"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
```

The *-m* flag enables you to find out which machine is the master server for a particular map. Suppose, for example, that you want to find out which machine is the master server for the *rpc* map you're using. Simply invoke *ypwhich* with the *-m* switch and the name of the map:

```
% ypwhich -m rpc
convexs
```

In this example, *convexs* is the master server for the the *rpc* map.

To locate master servers for maps in other domains, use the *-d* switch in conjunction with the *-m* switch:

```
% ypwhich -d eclipse -m networks
convex1
```

The *-V1* and *-V2* flags allow you to determine, respectively, which server is supplying Version 1.0 and Version 2.0 of the yellow pages. (Version 1.0 is provided for backward compatibility with older systems from other vendors. Version 2.0 is used as the default CONVEX version of *yp*.) For example:

```
% ypwhich -V1
convexs
% ypwhich -V2
convexs
```

You can print a list of every map in a domain (with its associated master server) by invoking the *-m* flag without arguments. For example:

```
% ypwhich -m
bootparams convexs
mail.aliases convexs
pwrestrict.byuid convexs
pwrestrict.byname convexs
netgroup.byhost convexs
netgroup.byuser convexs
netgroup convexs
protocols.byname convexs
services convexs
services.byname convexs
rpc.byname convexs
rpc.bynumber convexs
networks.byaddr convexs
networks.byname convexs
ethers.byname convexs
ethers.byaddr convexs
hosts.byaddr convexs
hosts.byname convexs
group.bygid convexs
group.byname convexs
passwd.byuid convexs
protocols.bynumber convexs
ypservers convexs
passwd.byname convexs
```

To list maps in other domains, use the *-d* switch in conjunction with the *-m* switch:

```
% ypwhich -d eclipse -m
bootparams eclipse
mail.aliases eclipse
netgroup.byhost eclipse
netgroup.byuser eclipse
netgroup eclipse
pwrestrict.byuid eclipse
pwrestrict.byname eclipse
protocols.byname eclipse
services eclipse
services.byname eclipse
rpc.byname eclipse
rpc.bynumber eclipse
networks.byaddr eclipse
networks.byname eclipse
ethers.byname eclipse
ethers.byaddr eclipse
```

```
hosts.byaddr eclipse
hosts.byname eclipse
group.bygid eclipse
group.byname eclipse
passwd.byuid eclipse
protocols.bynumber eclipse
ypservers eclipse
passwd.byname eclipse
```

ypclnt

ypclnt(3N) is a package of library functions that you can use to build your own interfaces to the yellow pages. These functions are designed to be used in *programs*; they are not stand-alone programs, and do not do you much good if you are not a C programmer. This section describes the available functions and provides examples. See the *ypclnt*(3N) manual page for more information and for a look at the functions themselves.

The functions available to you include the following:

yp_bind	Binds client program to a given *yp* server (domain)
yp_unbind	Unbinds client programs from given *yp* domain
yp_get_default_domain	Returns default system domain name
yp_match	Matches a key in a given *yp* map
yp_first	Returns first entry in a *yp* map
yp_next	Returns next entry in a *yp* map
yp_all	Transfers an entire map in one TCP request
yp_order	Returns map order number (a datestamp in seconds)
yp_master	Returns machine name of master server for a given map
yperr_string	Returns string describing a given error
ypprot_err	Converts *yp* protocol error to *ypclnt* error code

The examples shown in Figures 3-1, 3-2, and 3-3 illustrate the use of *ypclnt* functions.

Figure 3–1: Example 1—Using *ypclnt*

```
1       #include <stdio.h>
2       #include <rpcsvc/ypclnt.h>
3
4       #define MAP "passwd.byname"
5
6       main()
7       {
8               char *domain;
9               int err;
10              char *key, *old_key, *val;
11              int keylen, old_keylen, vallen;
12
13              if ((err = yp_get_default_domain(&domain)) != 0) {
14                      fprintf(stderr, "demo: %s\n", yperr_string(err));
15                      exit(1);
16              }
17              if ((err = yp_bind(domain)) != 0) {
18                      fprintf(stderr, "demo: %s\n", yperr_string(err));
19                      exit(1);
20              }
21              if ((err = yp_first(domain, MAP, &old_key, &old_keylen,
22                 &val, &vallen)) != 0) {
23                      fprintf(stderr, "demo: %s\n", yperr_string(err));
24                      exit(1);
25              }
26              printf("%s", val);
27              while (yp_next(domain, MAP, old_key, old_keylen, &key, &keylen,
28                 &val, &vallen) == 0) {
29                      printf("%s", val);
30                      old_key = key;
31                      old_keylen = keylen;
32              }
33              exit(0);
34      }
```

Example 1 shows the use of *ypclnt* functions *yp_get_default_domain* (at line 13); *yperr_string* (at line 14); *yp_bind* (at line 17); *yp_first* (at line 21); and *yp_next* (at line 27). The program finds the given domain, binds to it, and then reads and prints the entries in the file MAP. In function, this program is essentially a smaller version of *ypcat*.

Figure 3–2: Example 2—Using *ypclnt*

```
1       #include <stdio.h>
2       #include <rpc/rpc.h>
3       #include <rpcsvc/ypclnt.h>
4       #include <rpcsvc/yp_prot.h>
5
6       #define MAP "passwd.byname"
7
8       extern int map_foreach();
9
10      main()
11      {
12              char *domain;
13              int err;
14              char *key, *old_key, *val;
15              int keylen, old_keylen, vallen;
16              struct ypall_callback callback;
17
18              if ((err = yp_get_default_domain(&domain)) != 0) {
19                      fprintf(stderr, "demo: %s\n", yperr_string(err));
20                      exit(1);
21              }
22              if ((err = yp_bind(domain)) != 0) {
23                      fprintf(stderr, "demo: %s\n", yperr_string(err));
24                      exit(1);
25              }
26              callback.foreach = map_foreach;
27              if ((err = yp_all(domain, MAP, &callback)) != 0) {
28                      fprintf(stderr, "demo: %s\n", yperr_string(err));
29                      exit(1);
30              }
31              exit(0);
32      }
33
34      map_foreach(instatus, inkey, inkeylen, inval, invallen, indata)
35              int instatus;
36              char *inkey;
37              int inkeylen;
38              char *inval;
39              int invallen;
40              char *indata;
41      {
42              if (instatus != YP_TRUE) {
43                      return(1);
44              }
45              inkey[inkeylen] = inval[invallen] = '\0';
46              printf("%s\n", inval);
47              return(0);
48      }
```

Example 2 is a rewritten version of Example 1, in which *yp_all*, rather than *yp_first* and *yp_next*, is used. *yp_all* typically works much faster than the *yp_first*/*yp_next* combination, especially when it is likely that you need to read an entire map. When the code in Examples 1 and 2 were run across a sample *passwd* file, the code in Example 2 ran 500% faster. (Specifics: *passwd* file contained 182 entries and 14135 characters; time to read/print with code in Example 1 = 10 seconds, while code in Example 2 read and printed the file in 2 seconds.)

Note that in Example 2 the *inkey* and *inval* entries are not null-terminated. You are given the lengths of each string as *inkeylen* and *invallen*—you should null-terminate *these* strings, as shown at line 45.

Figure 3–3: Example 3—Using *ypclnt*

```
1       #include <sys/types.h>
2       #include <sys/time.h>
3       #include <stdio.h>
4       #include <rpcsvc/ypclnt.h>
5
6       #define MAP "passwd.byname"
7       #define KEY "daemon"
8
9       main()
10      {
11              char *domain, *master;
12              int err;
13              char *key, *val;
14              int keylen, vallen;
15              int order;
16
17              if ((err = yp_get_default_domain(&domain)) != 0) {
18                      fprintf(stderr, "demo: %s\n", yperr_string(err));
19                      exit(1);
20              }
21              if ((err = yp_bind(domain)) != 0) {
22                      fprintf(stderr, "demo: %s\n", yperr_string(err));
23                      exit(1);
24              }
25              if ((err = yp_match(domain, MAP, KEY, strlen(KEY), &val, &vallen)) != 0) {
26                      fprintf(stderr, "demo: %s\n", yperr_string(err));
27                      exit(1);
28              }
29              printf("%s", val);
30              if ((err = yp_order(domain, MAP, &order)) != 0) {
31                      fprintf(stderr, "demo: %s\n", yperr_string(err));
32                      exit(1);
33              }
34              printf("Map %s has order number %d -- %s", MAP, order, ctime(&order));
35              if ((err = yp_master(domain, MAP, &master)) != 0) {
36                      fprintf(stderr, "demo: %s\n", yperr_string(err));
37                      exit(1);
38              }
39              printf("The master server for map %s is %s\n", MAP, master);
40              exit(0);
41      }
42
43
44
```

Example 3 shows the use of *yp_match* (at line 25); *yp_order* (at line 30); and *yp_master* (at line 35). The program binds to a specified domain, and then lists values in MAP that match the KEY argument. *yp_match* is used, of course, to search MAP for a particular KEY. *yp_order* and *yp_master* are typically not used for application programs. They return the time that a map was built and the name of the machine that is the master server for MAP, respectively. The output for Example 3 is shown here:

```
daemon:6c2Lz36Q10Mc.:1:1:Our friend, the daemon:/:/bin/csh
Map passwd.byname has order number 532161701 -- Wed Nov 12 00:41:47 1986
The master server for map passwd.byname is convexs
```

A

Reporting Problems

Introduction

The *contact* utility is the recommended way to report software and documentation problems to the Technical Assistance Center (TAC). It is an interactive tool that prompts you for the information necessary to report a problem to the TAC.

You must have a UNIX-to-UNIX Communications Protocol (UUCP) connection to the TAC to use *contact*. A UUCP system allows communication between UNIX systems by either dial-up or hard-wired communication lines. See *uucp*(1) or the entry in *info*(1) (online information system) for more information.

You must know the name and version number of the product involved. If you do not know the version number of the program or utility you are having trouble with, use the *vers* command. The syntax for the command is

vers *filename*

where *filename* is the the full pathname of the program. If you don't know the full pathname of the program, type

which *program*

For more information on these commands, see *vers*(1) and *which*(1) in the *CONVEX UNIX Programmer's Manual*, Part I.

Information Required to Report a Problem

contact requires the following information:

1. Your name, title, phone number, and corporate name.
2. The name and version of the product involved. Use the *vers* command if you don't know the version number of the program or utility.
3. A short (1 line) summary of the problem.
4. A detailed description of the problem. Include source code and a stack backtrace whenever possible. (See *adb*(1) or *csd*(1) for information on obtaining stack backtraces.) The more information provided, the quicker your problem can be isolated and solved.
5. The priority of the problem. You are shown a list of six levels from which to select.
6. Instructions on how to reproduce the problem, including the command syntax used, any flags invoked, or anything else you attempted to make your program run.

7. Any other comments about the problem or files you wish to submit.

You will have a chance to review your report before you submit it. You can edit the report if you find an error in what you have typed. If you change your mind and don't want to submit the report, you can abort the *contact* session; the file is saved in your home directory in a file named *dead.report*.

The following figure is a sample *contact* session. User input is in bold lettering, and the system response is in constant-width lettering.

Figure A–1: Sample *contact* Session

```
%contact (RETURN)
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
> Margaret Atwood, systems programmer, 814-4444, University r
> of Chicago (RETURN)
> (CTRL-D)

Enter the name of the product involved
> CONVEX UNIX Programmer's Manual, Part I (RETURN)

Enter the version number (in the form X.X or X.X.X.X) of the product
> Revision 4.0 (RETURN)

Enter a short (1 line) summary of the problem
> The finger command manual page lists nonexistent bug (RETURN)

Enter a detailed description of the problem (^D to terminate)
> The finger(1) man page says, under the BUGS section, that "Only the first
line of the .project file is printed."  Happily, this is not true! (RETURN)
> (CTRL-D)

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.
> 4 (RETURN)

Enter the instructions by which the problem may be reproduced (^D to terminate)
> a) put more than one line in .project (RETURN)
> b) read the man page for finger(1) (RETURN)
> (CTRL-D)

Enter any comments that are applicable (^D to terminate) (RETURN)
> (CTRL-D)

Do you have any suggestions or comments on the documentation that you
referenced when you were trying to resolve your problem (for example,
additions, corrections organization, accessibility)? (^D to terminate)
> The man page should be updated. (RETURN)
> (CTRL-D)

Are there any files that should be included in this report (yes | no)?
> no (RETURN)

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.
> 3 (RETURN)

Problem report submitted.
%
```

Index

Software Documentation

Index Enhancements

So that we can continue to provide better indexing in CONVEX documentation, please keep track of the words or phrases you look up in an index, but don't find. Then, list under which index entry you ultimately found the information you were seeking. You can mail one of these postage-paid forms to the CONVEX Software Documentation Department monthly, or you can submit the information to the Technical Assistance Center in the form of a bug report. You can get more forms by writing to CONVEX at the address below, or by calling us. You can also photocopy this form and mail it back in an envelope. Thank you for helping us to serve you better.

Name:_____ **Company:**_____

Phone:_____ **Date:**_____

Manual Title/Rev. No.	Looked Up This Word	Found Information Under This Word

CONVEX Computer Corporation — P.O. Box 833851, Richardson, TX 75083-3851 (214)952-0200

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CUSTOMER SERVICE
CONVEX Computer Corp.
P.O. Box 833851
Richardson, TX 75083-3851

(Fold Here Second)

(Tape or Staple)

CONVEX Network File User's Guide
Document No. 710-001530-203, Second Edition, Rev. 2.

Reader's Forum

You are invited to submit your comments concerning the clarity and service of this manual. Constructive critical comments are most welcome and will help us continue in our efforts to generate quality customer documentation. (Please list the page number for questions and comments):

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION/DOCUMENTATION:

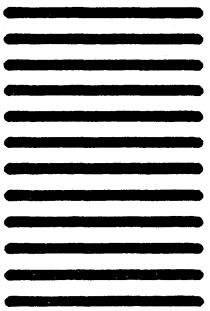
Within continental U.S.	1(800)952-0379
Outside continental U.S.	Contact local CONVEX office

Direct Mail Orders to

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

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CUSTOMER SERVICE
CONVEX Computer Corp.
P.O. Box 833851
Richardson, TX 75083-3851

(Fold Here Second)

(Tape or Staple)