IBM

eserver

iSeries

OS/400 Network File System Support

Version 5

SC41-5714-02

IBM

@server

iSeries

OS/400 Network File System Support

Version 5

SC41-5714-02

Note

Before using this information and the product it supports, be sure to read the information in "Notices" on page 101.

Third Edition (September 2002)

This edition replaces SC41-5714-01.

Contents

Figures

Tables

About OS/400 Network File System Support (SC41-5714)

The purpose of this book is to explain what the Network File System is, what it does, and how it works on the iSeries server. The book shows real-world examples of how you can use NFS to create a secure, useful integrated file system network. The intended audiences for this book are:

- System administrators developing a distributed network using the Network File System.
- Users or programmers working with the Network File System

Chapters one and two introduce NFS by giving background and conceptual information on its protocol, components, and architecture. This is background information for users who understand how the iSeries server works, but do not understand NFS.

The rest of the book (chapters three through nine) shows detailed examples of what NFS can do and how you can best use it. The overall discussion topic of this book is how to construct a secure, user-friendly distributed namespace. Included are in-depth examples and information regarding mounting, exporting, and the following topics:

- How NFS functions in the client/server relationship
- NFS exceptions for iSeries file systems
- NFS startup, shutdown, and recovery
- File locking
- New integrated file system error conditions and how NFS affects them
- Troubleshooting procedures for NFS security considerations

It is assumed that the reader has experience with iSeries client/server model, though not necessarily with the Network File System.

Who should read this book

This book is for iSeries users, programmers, and administrators who want to know about the Network File System on the iSeries server. This book contains:

- Background theory and concepts regarding NFS and how it functions
- Examples of commands, iSeries displays, and other operations you can use with NFS
- Techniques on how to construct a secure, efficient namespace with NFS

Prerequisite and related information

Use the iSeries Information Center as your starting point for looking up iSeries technical information.

You can access the Information Center two ways:

- From the following Web site:

  `http://www.ibm.com/eserver/iseries/infocenter`
- From CD-ROMs that ship with your Operating System/400 order:

iSeries Information Center, SK3T-4091-02. This package also includes the PDF versions of iSeries manuals, *iSeries Information Center: Supplemental Manuals*, SK3T-4092-01, which replaces the Softcopy Library CD-ROM.

The iSeries Information Center contains advisors and important topics such as Java, TCP/IP, Web serving, secured networks, logical partitions, clustering, CL commands, and system application programming interfaces (APIs). It also includes links to related IBM Redbooks and Internet links to other IBM Web sites such as the Technical Studio and the IBM home page.

With every new hardware order, you receive the *iSeries Setup and Operations CD-ROM*, SK3T-4098-01. This CD-ROM contains IBM **@server** iSeries Access for Windows and the EZ-Setup wizard. iSeries Access offers a powerful set of client and server capabilities for connecting PCs to iSeries servers. The EZ-Setup wizard automates many of the iSeries setup tasks.

For other related information, see the "Bibliography" on page 105.

iSeries Navigator

IBM iSeries Navigator is a powerful graphical interface for managing your iSeries servers. iSeries Navigator functionality includes system navigation, configuration, planning capabilities, and online help to guide you through your tasks. iSeries Navigator makes operation and administration of the server easier and more productive and is the only user interface to the new, advanced features of the OS/400 operating system. It also includes Management Central for managing multiple servers from a central server.

You can find more information on iSeries Navigator in the iSeries Information Center and at the following Web site:

`http://www.ibm.com/eserver/iseries/navigator/`

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other iSeries documentation, fill out the readers' comment form at the back of this book.

- If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
  - United States, Canada, and Puerto Rico: 1-800-937-3430
  - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use one of these e-mail addresses:
  - Comments on books:

    RCHCLERK@us.ibm.com
  - Comments on the iSeries Information Center:

    RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book or iSeries Information Center topic.
- The publication number of a book.
- The page number or topic of a book to which your comment applies.

Chapter 1. What is the Network File System?

Introduction

OS/400 Network File System Support introduces a system function for the iSeries server that aids users and administrators who work with network applications and file systems. You can use the Network File System (NFS**) to construct a distributed network system where all users can access the data they need. Furthermore, the Network File System provides a method of transmitting data in a client/server relationship.

The Network File System makes remote objects stored in file systems appear to be local, as if they reside in the local host. With NFS, all the servers in a network can share a single set of files. This eliminates the need for duplicate file copies on every network system. Using NFS aids in the overall administration and management of users, systems, and data.

NFS gives users and administrators the ability to distribute data across a network by:

- **Exporting** local file systems from a local server for access by remote clients. This allows centralized administration of file system information. Instead of duplicating common directories on every server, NFS shares a single copy of a directory with all the proper clients from a single server.
- **Mounting** remote server file systems over local client directories. This allows iSeries servers to work with file systems that have been exported from a remote server. The mounted file systems will act and perform as if they exist on the local server.

The following figures show the process of a remote NFS server exporting directories to a local client. Once the client is aware of the exported directories, the client then mounts the directories over local directories. The remote server directories will now function locally on the client.

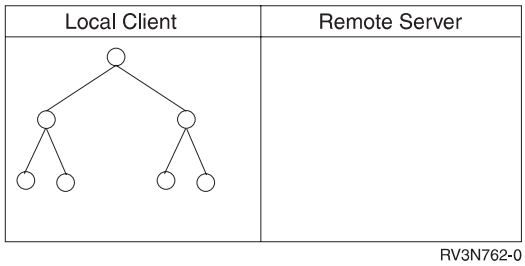

Figure 1. The local client and its view of the remote server before exporting data

Before the server **exports** information, the client does not know about the existence of file systems on the server. Furthermore, the client does not know about any of the file systems or objects on the server.

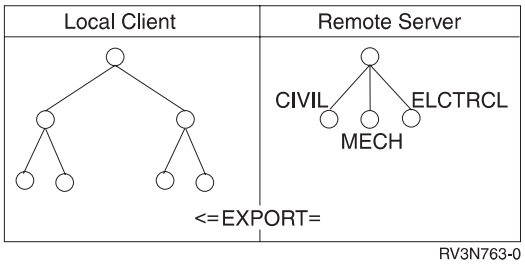

Figure 2. The local client and its view of the remote server after exporting data

After the server exports information, the proper client (the client with the proper authorities) can be aware of the existence of file systems on the server. Furthermore, the client can **mount** the exported file systems or directories or objects from the server.

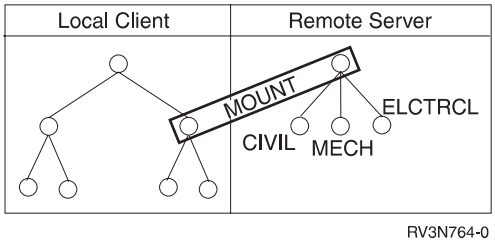

Figure 3. The local client mounts data from a remote server

The mount command makes a certain file system, directory, or object *accessible* on the client. Mounting does not copy or move objects from the server to the client. Rather, it makes *remote* objects available for use *locally*.

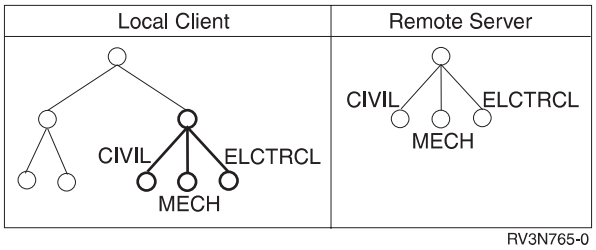

Figure 4. Remote file systems function on the client

When remote objects are mounted locally, they *cover up* any local objects that they are placed over. Mounted objects also cover any objects that are *downstream* of the **mount point**, the place on the client where the mount to the server begins. The mounted objects will function locally on the client just as they do remotely on the server.

For more information on these aspects of NFS, see the following sections:
- Chapter 4, “Server Exporting of File Systems” on page 27
- Chapter 5, “Client Mounting of File Systems” on page 43

NFS is the replacement for the TCP/IP File Server Support/400 (FSS/400) system application. If you are accustomed to working with FSS/400, you will notice many

similarities between FSS/400 and NFS. It is important to note, however, that FSS/400 and NFS are ***not*** compatible with each other. FSS/400 can exist on the same server with NFS, but they cannot operate together. Do not start or use FSS/400 and NFS at the same time.

A Brief History

Sun Microsystems, Inc.** released NFS in 1984. Sun introduced NFS Version 2 in 1985. In 1989, the Request For Comments (RFC) standard 1094, describing NFS Version 2, was published. X/Open published a compatible version that is a standard for NFS in 1992. Sun published the NFS Version 3 protocol in 1993.

Sun developed NFS in a UNIX** environment; therefore, many UNIX® concepts (for example, the UNIX authentication) were integrated into the final protocol. Even so, NFS is platform independent. Today, almost all UNIX platforms use NFS, as do many PCs, mainframes, and workstations. The iSeries server implementation of NFS supports Version 2 and Version 3 of the protocol.

The Network File System as a File System

The iSeries' file systems provide the support that allows users and applications to access specific segments of storage. These logical units of storage are the following:
- libraries
- directories
- folders

The logical storage units can contain different types of data:
- objects
- files
- documents

Each file system has a set of logical structures and rules for interacting with information in storage. These structures and rules may be different from one file system to another, depending on the type of file system. The OS/400 support for accessing database files and various other object types through libraries can be thought of as a file system. Similarly, the OS/400 support for accessing documents through folders can be thought of as a separate file system. For more information on iSeries file systems, please see File systems in the integrated file system in the **File systems and management** topic of the iSeries Information Center.

The Network File System provides seemingly "transparent" access to remote files. This means that local client files and files that are accessed from a remote server operate and function similarly and are indistinguishable. This takes away many complex steps from users, who need a set of files and directories that act in a consistent manner across many network clients. A long-term goal of system administrators is to design such a transparent network that solidifies the belief of users that all data exists and is processed on their local workstations. An efficient NFS network also gives the right people access to the right amount of data at the right times.

Files and directories can be made available to clients by **exporting** from the server and **mounting** on clients through a pervasive NFS client/server relationship. An NFS client can also, at the same time, function as an NFS server, just as an NFS server can function as a client.

Stateless Network Protocol

NFS incorporates the Remote Procedure Call (RPC) for client/server communication. RPC is a high-end network protocol that encompasses many simpler protocols, such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

NFS is a **stateless protocol**, maintaining absolutely no saved or archived information from client/server communications. State is the information regarding a request that describes exactly what the request does. A stateful protocol saves information about users and requests for use with many procedures. Statelessness is a condition where no information is retained about users and their requests. This condition demands that the information surrounding a request be sent with every single request. Due to NFS statelessness, each RPC request contains all the required information for the client and server to process user requests.

By using NFS, you can bypass the details of the network interface. NFS isolates applications from the physical and logical elements of data communications and allows applications to use a variety of different transports.

In short, the NFS protocol is useful for applications that need to transfer information over a client/server network. For more information about RPC and NFS, see "Network File System Stack Description" on page 8.

Overview of the TULAB Scenario

This book uses the fictional namespace **TULAB** to describe detailed applications of NFS concepts. A **namespace** is a distributed network space where one or more servers look up, manage, and share ordered, deliberate object names.

TULAB exists only in a hypothetical computer-networked environment at a fictitious Technological University. It is run by a **network administrator**, a person who defines the network configuration and other network-related information. This person controls how an enterprise or system uses its network resources. The TULAB administrator, Chris Admin, is trying to construct an efficient, transparent, and otherwise seamless distributed namespace for the diverse people who use the TULAB:

- Engineering undergraduate students
- Humanities undergraduate students
- Engineering graduate students
- TULAB consultants

Each group of users works on sets of clients that need different file systems from the TULAB server. Each group of users has different permissions and authorities and will pose a challenge to establishing a safe, secure NFS namespace.

Chris Admin will encounter common problems that administrators of NFS namespaces face every day. Chris Admin will also work through some uncommon and unique NFS situations. As this book describes each command and parameter, Chris Admin will give a corresponding example from TULAB. As this book explains applications of NFS, Chris Admin will show exactly how he configures NFS for TULAB.

The network namespace of TULAB is complex and involves two NFS server systems:

1. **TULAB1** — A UNIX server system

2. **TULAB2** — An iSeries server system

The following figure describes the layout of the TULAB namespace.

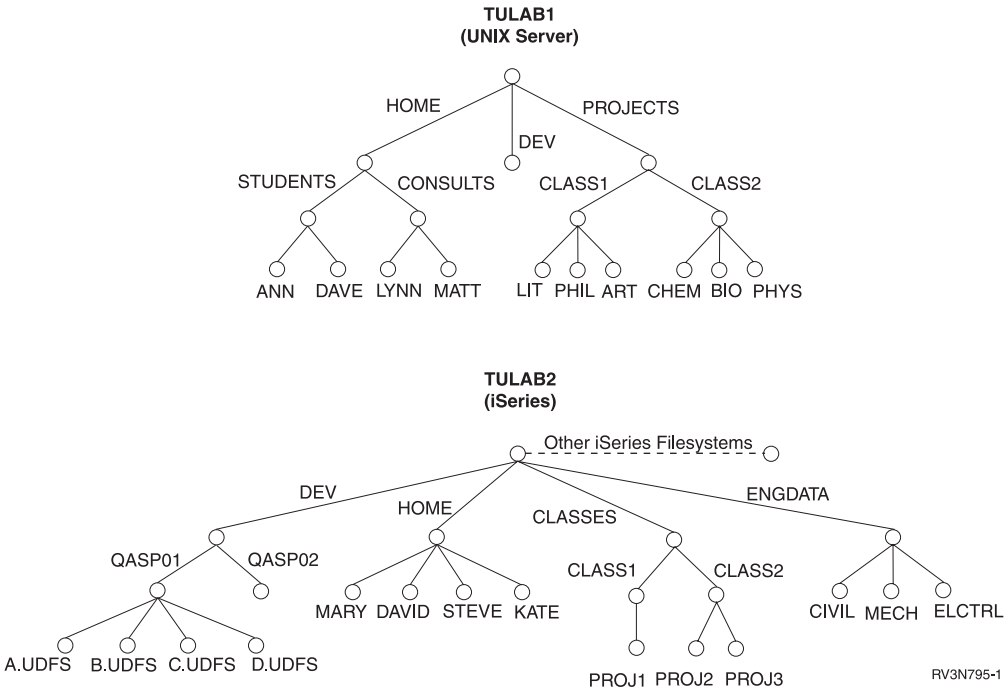

Figure 5. The TULAB network namespace

Chapter 2. The Network File System Client/Server Model

To understand how the Network File System works on the iSeries server, you must first understand the communication relationship between a server and various clients. The **client/server model** involves a local host (the **client**) that makes a procedure call that is usually processed on a different, remote network system (the **server**). To the client, the procedure appears to be a local one, even though another system processes the request. In some cases, however, a single computer can act as both an NFS client *and* an NFS server.

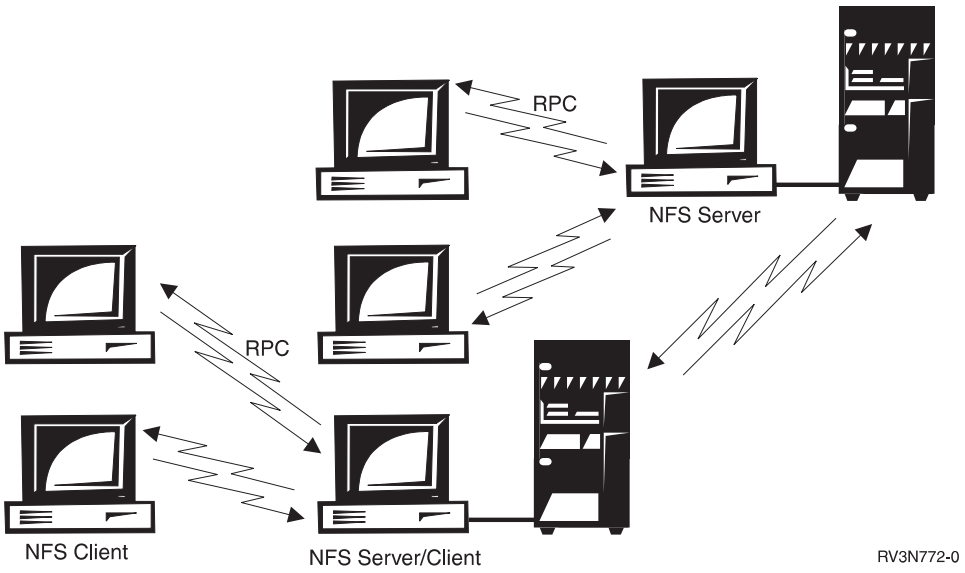

Figure 6. The NFS Client/Server Model

There are various resources on the server that are not available on the client, hence the need for such a communication relationship. The host owning the needed resource acts as a server that communicates to the host, which initiates the original call for the resource, the client. In the case of NFS, this resource is usually a shared file system, a directory, or an object.

RPC is the mechanism for establishing such a client/server relationship within NFS. RPC bundles up the arguments intended for a procedure call into a packet of data called a network datagram. The NFS client creates an RPC session with an NFS server by connecting to the proper server for the job and transmitting the datagram to that server. The arguments are then unpacked and decoded on the server. The operation is processed by the server, and a return message (should one exist) is sent back to the client. On the client, this reply is transformed into a return value for NFS. The user's application is re-entered as if the process had taken place on a local level.

Network File System Client/Server Communication Design

The logical layout of the Network File System on the client and server involves numerous daemons, caches, and the NFS protocol breakdown. An overview of each type of process follows.

A **daemon** is a process that performs continuous or system-wide functions, such as network control. NFS uses many different types of daemons to complete user requests.

A **cache** is a type of high-speed buffer storage that contains frequently accessed instructions and data. Caches are used to reduce the access time for this information. **Caching** is the act of writing data to a cache.

For information about NFS server daemons, see "Network File System Server-Side Daemons" on page 9. For information about NFS client daemons, see "Network File System Client-Side Daemons" on page 12. For information about client-side caches, see "NFS Client-Side Caches" on page 12 Detailed information about the NFS protocol can be found in "Network File System Stack Description".

Network File System Process Layout

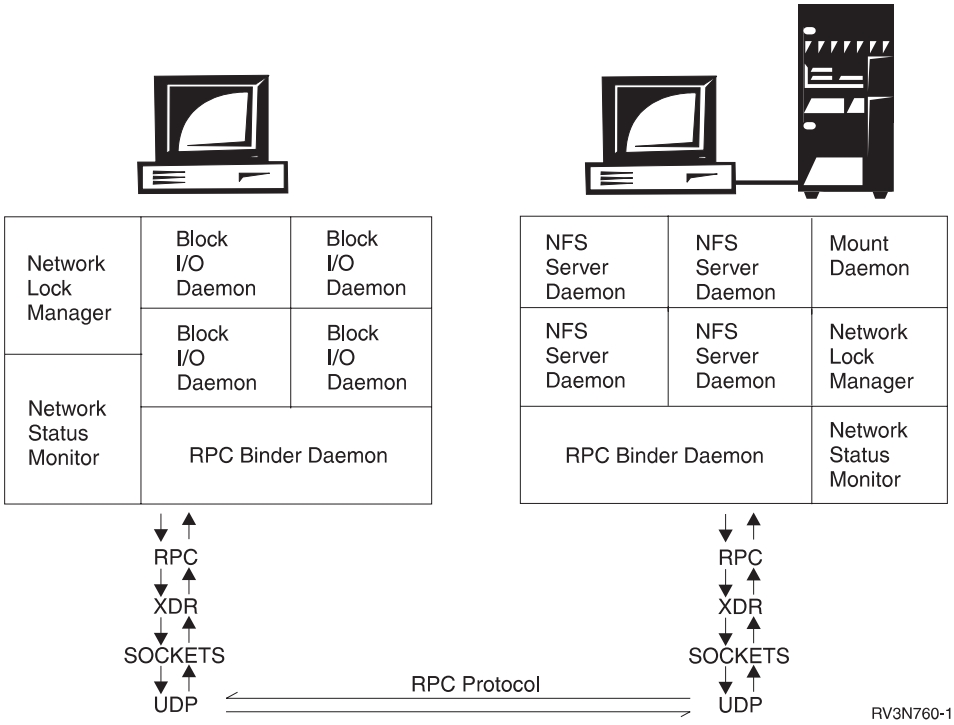

Figure 7. A breakdown of the NFS client/server protocol

Local processes that are known as **daemons** are required on both the client and the server. These daemons process both local and remote requests and handle client/server communication. Both the NFS client and server have a set of daemons that carry out user tasks. In addition, the NFS client also has data caches that store specific types of data locally on the on the client. For more information about the NFS client data caches, see "NFS Client-Side Caches" on page 12.

Network File System Stack Description

Simple low-end protocols make up a high-end complex protocol like NFS. For an NFS client command to connect with the server, it must first use the Remote Procedure Call (RPC) protocol. The request is encoded into External Data Representation (XDR) and then sent to the server using a socket. The simple User

Datagram Packet (UDP) protocol actually communicates between client and server. Some aspects of NFS use the Transmission Control Protocol (TCP) as the base communication protocol.

The operation of NFS can be seen as a logical client-to-server communications system that specifically supports network applications. The typical NFS flow includes the following steps:

1. The server waits for requests from one or more clients.
2. The client sends a request to the server and blocks (waits for a response).
3. When a request arrives, the server calls a dispatch routine.
4. The dispatch routine performs the requested service and returns with the results of the request. The dispatch routine can also call a sub-routine to handle the specific request. Sometimes the sub-routine will return results to the client by itself, and other times it will report back to the dispatch routine.
5. The server sends those results back to the client.
6. The client then de-blocks.

The overhead of running more than one request at the same time is too heavy for an NFS server, so it is designed to be *single-threaded*. This means that an NFS server can only process one request per session. The requests from the multiple clients that use the NFS server are put into a queue and processed in the order in which they were received. To improve throughput, multiple NFS servers can process requests from the same queue.

iSeries as a Network File System Server

The NFS server is composed of many separate entities that work together to process remote calls and local requests. These are:

- **NFS server daemons**. These daemons handle access requests for local files from remote clients. Multiple instances of particular daemons can operate simultaneously.
- **Export command**. This command allows a user to make local directories accessible to remote clients.
- **`/etc/exports` file**. This file contains the local directory names that the NFS server exports automatically when starting up. The administrator creates and maintains this file, which is read by the export command. For more information about the `/etc/exports` file, see "/etc/exports File" on page 96 and Chapter 4, "Server Exporting of File Systems" on page 27.
- **Export table**. This table contains all the file systems that are currently exported from the server. The export command builds the `/etc/exports` file into the export table. You can dynamically update the export table with the export command.

  For more information about the Change Network File System Export (CHGNFSEXP) and Export File System (EXPORTFS) commands and how they work with the `/etc/exports` file, see Chapter 4, "Server Exporting of File Systems" on page 27.

Network File System Server-Side Daemons

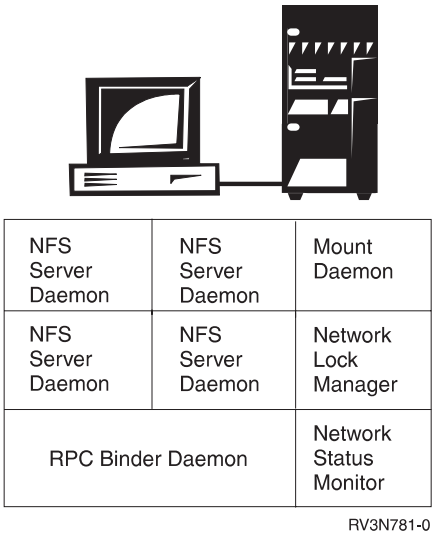

Figure 8. The NFS Server

NFS is similar to other RPC-based services in its use of server-side daemons to process incoming requests. NFS may also use multiple copies of some daemons to improve overall performance and efficiency.

RPC Binder Daemon

The RPC binder daemon is analogous to the port mapper daemon, which many implementations of NFS use in UNIX. Clients determine the port of a specified RPC service by using the RPC binder daemon. Local services register themselves with the local RPC binder daemon (port mapper) when initializing. On iSeries, you can register your own RPC programs with the RPC binder daemon.

NFS Server Daemons

The most pressing need for NFS server daemons centers around the need for multi-threading NFS RPC requests. Running daemons in user-level processes allows the server to have multiple, independent threads of processes. In this way, the server can handle several NFS requests at once. As a daemon completes the processing of a request, the daemon returns to the end of a line of daemons that wait for new requests. Using this schedule design, a server always has the ability to accept new requests if at least one server daemon is waiting in the queue. Multiple instances of this daemon can perform tasks simultaneously.

Mount Daemon

Each NFS server system runs a mount daemon which listens to requests from client systems. This daemon acts on mount and unmount requests from clients. If the mount daemon receives a client mount request, then the daemon checks the export table. The mount daemon compares it with the mount request to see if the client is allowed to perform the mount. If the mount is allowed, the mount daemon will send to the requesting client an opaque data structure, the file handle. This structure uniquely describes the mounting point that is requested by the client. This will enable the client to represent the root of the mounted file system when making future requests.

Network Status Monitor Daemon

The Network Status Monitor (NSM) is a **stateful** NFS service that provides applications with information about the status of network hosts. The Network Lock Manager (NLM) daemon heavily uses the NSM to track hosts that have established locks as well as hosts that maintain such locks.

There is a single NSM server per host. It keeps track of the state of clients and notifies any interested party when this state changes (usually after recovery from a crash).

The NSM daemon keeps a *notify list* that contains information on hosts to be informed after a state change. After a local change of state, the NSM notifies each host in the notify list of the new state of the local NSM. When the NSM receives a state change notification from another host, it will notify the local network lock manager daemon of the state change.

Network Lock Manager Daemon

The Network Lock Manager (NLM) daemon is a **stateful** service that provides advisory byte-range locking for NFS files. The NLM maintains state across requests, and makes use of the Network Status Monitor daemon (NSM) which maintains state across crashes (using stable storage).

The NLM supports two types of byte-range locks:

1. **Monitored locks**. These are reliable and helpful in the event of system failure. When an NLM server crashes and recovers, all the locks it had maintained will be reinstated without client intervention. Likewise, NLM servers will release all old locks when a client crashes and recovers. A Network Status Manager (NSM) must be functioning on both the client and the server to create monitored locks.
2. **Unmonitored locks**. These locks require explicit action to be released after a crash and re-established after startup. This is an alternative to monitoring locks, which requires the NSM on both the client and the server systems.

iSeries as a Network File System Client

Several entities work together to communicate with the server and local jobs on the NFS client. These processes are the following:

- **RPC Binder Daemon**. This daemon communicates with the local and remote daemons by using the RPC protocol. Clients look for NFS services through this daemon.
- **Network Status Monitor** and **Network Lock Manager**. These two daemons are not mandatory on the client. Many client applications, however, establish byte-range locks on parts of remote files on behalf of the client without notifying the user. For this reason, it is recommended that the NSM and NLM daemons exist on both the NFS client and server.
- **Block I/O daemon**. This daemon manages the data caches and is therefore stateful in operation. It performs caching, and assists in routing client-side NFS requests to the remote NFS server. Multiple instances of this daemon can perform tasks simultaneously.
- **Data and attribute caches**. These two caches enhance NFS performance by storing information on the client-side to prevent a client/server interaction. The attribute cache stores file and directory attribute information locally on the client, while the data cache stores frequently used data on the client.
- **Mount and Unmount commands**. Users can mount and unmount a file system in the client namespace with these commands. These are general tools, used not only in NFS, but also to dynamically mount and unmount other local file systems. For more information about the Add Mounted File System (ADDMFS) and Remove Mounted File System (RMVMFS) commands, see Chapter 5, “Client Mounting of File Systems” on page 43.

Network File System Client-Side Daemons

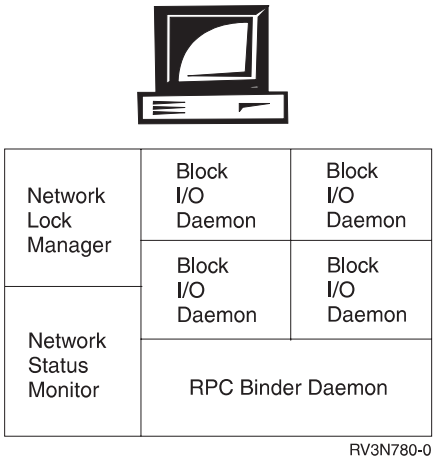

Figure 9. The NFS Client

Besides the RPC Daemon, the NFS client has only one daemon to process requests and to transfer data from and to the remote server, the block I/O daemon. NFS differs from typical client/server models in that processes on NFS clients make some RPC calls themselves, independently of the client block I/O daemon. An NFS client can optionally use both a Network Lock Manager (NLM) and a Network Status Monitor (NSM) locally, but these daemons are not required for standard operation. It is recommended that you use both the NLM and NSM on your client because user applications often establish byte-range locks without the knowledge of the user.

Block I/O Daemon

The block I/O daemon handles requests from the client for remote files or operations on the server. The block I/O daemon may handle data requests from the client to remote files on the server. Running only on NFS clients or servers that are also clients, this daemon manages the data caches for the user. The block I/O daemon is stateful and routes client application requests either to the caches or on to the NFS server. The user can specify the regular intervals for updating all data that is cached by the block I/O daemon. Users can start multiple daemons to perform different operations simultaneously.

NFS Client-Side Caches

Caching file data or attributes gives administrators a way of tuning NFS performance. The caching of information allows you to delay writes or to read ahead.

Client-side caching in NFS reduces the number of RPC requests sent to the server. The NFS client can cache data, which can be read out of local memory instead of from a remote disk. The caching scheme available for use depends on the file system being accessed. Some caching schemes are prohibited because they cannot guarantee the integrity and consistency of data that multiple clients simultaneously change and update. The standard NFS cache policies ensure that performance is acceptable while also preventing the introduction of state into the client/server communication relationship.

There are two types of client caches: the **directory and file attribute cache** and the **data cache**.

Directory and File Attribute Cache

Not all file system operations use the data in files and directories. Many operations get or set the attributes of the file or directory, such as its length, owner, and modification time. Because these attribute-only operations are frequent and do not affect the data in a file or directory, they are prime candidates for using cached information.

The file and directory cache on the client store file attributes. The server does this so that every operation that gets or sets attributes does not have to go through the connection to the NFS server. When the server reads a file's attributes, the attributes remain valid on the client for a minimum period of time, typically 30 seconds. You can set this time period by using the `acregmin` option on the MOUNT command. If the file is changed, the attributes are updated, which makes changes to the local copy of the attributes and extends the cache validity period for another minimum period of time.

The attributes of a file remain unchanged for some maximum period of time, typically 60 seconds. You can set this time period with the `acregmax` option on the MOUNT command. The server then deletes file attributes from the cache and writes changed file attributes back to the server.

To force a refresh of attributes when opening a file, use the `nocto` option on the MOUNT command. Specifying the `noac` option stops the local caching of attributes, negating the `acregmin`, `acregmax`, `acdirmin`, and the `acdirmax` options on the MOUNT command.

The same mechanism is used for directory attributes, although they are given a longer period of time to be valid. The minimum and maximum period of time for directory attributes in the cache is set by the `acdirmin` and `acdirmax` options on the MOUNT command.

Attribute caching allows you to make multiple changes to a file or directory without having to constantly get and set attributes from the server. Intermediate attributes are cached, and all updates are written to the server when the maximum attribute cache period expires. Frequently accessed files and directories have their attributes cached locally on the client so that some NFS requests can be performed without having to make an RPC call. By preventing this type of client/server interaction, caching attributes improves the performance of NFS.

For more information on the ADDMFS and MOUNT commands, see Chapter 5, "Client Mounting of File Systems" on page 43. See the CL topic in the iSeries Information Center for more information on the options to the ADDMFS and MOUNT commands.

Data Cache

The data cache is very similar to the directory and file attribute cache in that it stores frequently used information locally on the client. The data cache, however, stores data that is frequently or likely to be used instead of file or directory attributes. The data cache provides data in cases where the client would have to access the server to retrieve information that has already been read. This operation improves the performance of NFS.

Whenever a user makes a request on a remote object, a request is sent to the server. If the request is to read a small amount of data, for example, 1 byte (B), then the server returns 4 kilobytes (KB) of data. This "extra" data is stored in the client caches because, presumably, it will soon be read by the client.

When users access the same data frequently over a given period of time, the client can cache this information to prevent a client/server interaction. This caching also applies to users who use data in one “area” of a file frequently. This is called *locality* and involves not only the primary data that is retrieved from the server, but also a larger block of data around it. When a user requests data frequently from one area, the entire block of data is retrieved and then cached. There is a high probability that the user will soon want to access this surrounding data. Because this information is already cached locally on the client, the performance of NFS is improved.

Client Timeout

If the client does not have a cache loaded, then all requests will go to the server. This takes extra time to process each client operation. With the mount command, users can specify a timeout value for re-sending the command. The mount command can not distinguish between a slow server and a server that does not exist, so it will retry the command.

The default retry value is 2 seconds. If the server does not respond in this time, then the client will continue to retry the command. In a network environment, this can overload the server with duplicate iSeries client requests. The solution to this difficulty is to increase the timeout value on the mount command to 5-10 seconds.

Chapter 3. NFS and the User-Defined File System (UDFS)

A user-defined file system (UDFS) is a type of file system that you directly manage through the end user interface. This contrasts with a system-defined file system (SDFS), which iSeries server code creates. QDLS, QSYS.LIB, and QOPT are all examples of SDFSs.

The UDFS introduces a concept on iSeries that allows you to create and manage your own file systems on a particular user Auxiliary Storage Pool (ASP). An ASP is a storage unit that is defined from the disk units or disk unit sub-systems that make up auxiliary storage. ASPs provide a means for placing certain objects on specific disk units to prevent the loss of data due to disk media failures on other disk units.

The concept of Block Special Files (*BLKSF objects) allows a user to view a UDFS as a single entity whose contents become visible only after mounting the UDFS in the local namespace. An unmounted UDFS appears as a single, opaque entity to the user. Access to individual objects within a UDFS from the integrated file system interface is permissible only when the UDFS is mounted.

UDFS support enables you to choose which ASP will contain the file system, as well as manage file system attributes like case-sensitivity. You can export a mounted UDFS to NFS clients so that these clients can also share the data that is stored on your ASP. This chapter explains how to create and work with a UDFS so that it can be used through NFS.

User File System Management

The UDFS provides new file management strategies to the user and includes several new and changed CL commands specific to UDFSs.

For more information about the various UDFS CL commands and their associated parameters and options, see the CL topic in the iSeries Information Center.

Create a User-Defined File System

The Create User-Defined File System (CRTUDFS) command creates a file system whose contents can be made visible to the rest of the integrated file system namespace via the ADDMFS (Add Mounted File System) or MOUNT command. A UDFS is represented by the object type *BLKSF, or block special file. Users can create a UDFS in an ASP of their own choice and have the ability to specify case-sensitivity.

Restrictions:

1. You must have *IOSYSCFG special authority to use this command.

CRTUDFS Display

```
Create User-Defined FS (CRTUDFS)

 Type choices, press Enter.

 User-defined file system . . . . > '/DEV/QASP02/kate.udfs'


 Public authority for data  . . .   *INDIR   Name, *INDIR, *RWX, *RW...
 Public authority for object  . .   *INDIR   *INDIR, *NONE, *ALL...
                + for more values
 Auditing value for objects . . .   *SYSVAL   *SYSVAL, *NONE,
*USRPRF...
                           Additional Parameters

 Case sensitivity . . . . . .   *MIXED   *MIXED, *MONO
 Default file format  . . . . .   *TYPE2   *TYPE1, *TYPE2
 Text 'description' . . . . . .   *BLANK

                                                                      Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 10. Using the Create User-Defined FS (CRTUDFS) display

When you use the CRTUDFS command, you can specify many parameters and options:

- The required UDFS parameter determines the name of the new UDFS. This entry must be of the form /DEV/QASPXX/name.udfs, where the XX is one of the valid Auxiliary Storage Pool (ASP) numbers on the system, and name is the name of the user-defined file system. All other parts of the path name must appear as in the example above. The name part of the path must be unique within the specified QASPXX directory.
- The DTAAUT parameter on the CRTUDFS command specifies the public data authority given to the user for the new UDFS.
- The OBJAUT parameter on the CRTUDFS command specifies the public object authority given to users for the new UDFS.
- The CRTOBJAUD parameter on the CRTUDFS command specifies the auditing value of objects created in the new UDFS.
- The CASE parameter on the CRTUDFS command specifies the case-sensitivity of the new UDFS. You can specify either the *MONO value or the *MIXED value. Using the *MONO value creates a case-insensitive UDFS. Using the *MIXED value creates a case-sensitive UDFS.
- The DFTFILEFMT specifies the format of stream files (*STMF) created in the UDFS. You can specify either the *TYPE1 value or the *TYPE2 value. A *TYPE1 *STMF has the same value as *STMF objects in releases prior to version 4, release 4 of OS/400®.
- The TEXT parameter on the CRTUDFS command specifies the text description for the new UDFS.

Examples

Example 1: Create UDFS in System ASP on TULAB2

```
CRTUDFS UDFS('/DEV/QASP01/A.udfs) CASE(*MONO)
```

This command creates a case-insensitive user-defined file system (UDFS) named `A.udfs` in the system Auxiliary Storage Pool (ASP), `qasp01`.

Example 2: Create UDFS in user ASP on TULAB2

```
CRTUDFS  UDFS('/DEV/QASP02/kate.udfs') CASE(*MIXED)
```

This command creates a case sensitive user-defined file system (UDFS) named `kate.udfs` in the user Auxiliary Storage Pool (ASP), `qasp02`.

Display a User-Defined File System

The Display User-Defined File System (DSPUDFS) command presents the attributes of an existing UDFS, whether mounted or unmounted.

DSPUDFS Display

```
Display User-Defined FS (DSPUDFS)

 Type choices, press Enter.

 User-defined file system . . . .   /DEV/QASP02/kate.udfs
 Output . . . . . . . . . . . . .   *        *, *PRINT









                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 11. Display User-Defined FS (DSPUDFS) output (1/2)

When you use the DSPUDFS command, you only have to specify one parameter:

- The UDFS parameter determines the name of the UDFS to display. This entry must be (or resolve to a pathname) of the form `/DEV/QASP02/name.udfs`, where the `XX` is one of the valid user Auxiliary Storage Pool (ASP) numbers on the system, and name is the name of the UDFS. All other parts of the path name must appear as in the example above.

When you use the DSPUDFS command successfully, a screen will appear with information about your UDFS on it:

```
Display User-Defined FS

 User-defined file system . . . :   /DEV/QASP02/kate.udfs

 Owner  . . . . . . . . . . . . :   PATRICK
 Code page  . . . . . . . . . . :   37
 Case sensitivity . . . . . . . :   *MIXED
 Creation date/time . . . . . . :   02/26/96  08:00:00
 Change date/time . . . . . . . :   08/30/96  12:30:42
 Path where mounted . . . . . . :   Not mounted

 Description  . . . . . . . . . :

                                                                    Bottom
 Press Enter to continue.

 F3=Exit   F12=Cancel
 (C) COPYRIGHT IBM CORP. 1980, 1996.
```

Figure 12. Display User-Defined FS (DSPUDFS) output (2/2)

Example
Display UDFS in user ASP on TULAB2

```
DSPUDFS UDFS('/DEV/QASP02/kate.udfs')
```

This command displays the attributes of a user-defined file system (UDFS) named `kate.udfs` in the user Auxiliary Storage Pool (ASP), `qasp02`.

Delete a User-Defined File System

The Delete User-Defined File System command (DLTUDFS) deletes an existing, unmounted UDFS and all the objects within it. The command will fail if the UDFS is mounted. Deletion of a UDFS will cause the deletion of all objects in the UDFS. If the user does not have the necessary authority to delete any of the objects within a UDFS, none of the objects in the UDFS will be deleted.

Restrictions:

1. The UDFS being deleted must not be mounted.
2. Only a user with *IOSYSCFG special authority can use this command.

DLTUDFS Display

Figure 13. Using the Delete User-Defined FS (DLTUDFS) display

When you use the DLTUDFS command, you only have to specify one parameter:

- The UDFS parameter determines the name of the unmounted UDFS to delete.

This entry must be of the form `/DEV/QASPXX/name.udfs`, where the `XX` is one of the valid Auxiliary Storage Pool (ASP) numbers on the system, and name is the name of the UDFS. All other parts of the path name must appear as in the example above. Wildcard characters such as '*' and '?' are not allowed in this parameter. The command will fail if the UDFS specified is currently mounted.

Example

Unmount and Delete a UDFS in the user ASP on TULAB2

```
UNMOUNT TYPE(*UDFS) MFS('DEV/QASP02/kate.udfs')
```

This command will unmount the user-defined file system (UDFS) named `kate.udfs` from the integrated file system namespace. A user must unmount a UDFS before deleting it. After unmounting a UDFS, a user can now proceed to delete the UDFS and all objects within it using the DLTUDFS command:

```
DLTUDFS UDFS('/DEV/QASP02/kate.udfs')
```

This command deletes the user-defined file system (UDFS) named `kate.udfs` from the user Auxiliary Storage Pool (ASP) `qasp02`.

Mount a User-Defined File System

The Add Mounted File System (ADDMFS) and MOUNT commands make the objects in a file system accessible to the integrated file system namespace. To mount a UDFS, you need to specify TYPE (*UDFS) for the ADDMFS command.

The ADDMFS command (or its alias, MOUNT) allows you to dynamically mount a file system, whether that file system is UDFS, NFS, or NetWare. Use the following steps to allow a successful export of a UDFS to NFS clients:

1. Mount the block special file locally (Type *UDFS)
2. Export the path to the UDFS mount point (the directory you mounted over in Step 1)

The previous steps will ensure that the remote view of the namespace is the same as the local view. Afterwards, the exported UDFS file system can be mounted (Type *NFS) by remote NFS clients. However, you must have previously mounted it on the local namespace.

ADDMFS/MOUNT Display

For a display of the ADDMFS (Add Mounted File System) and MOUNT commands, please see "RMVMFS/UNMOUNT Display" on page 52.

Example

Mount and Export a UDFS on TULAB2

```
MOUNT TYPE(*UDFS) MFS('/DEV/QASP02/kate.udfs') MNTOVRDIR('/usr')
```

This command mounts the user-defined file system (UDFS) that is named `kate.udfs` on the integrated file system namespace of TULAB2 over directory `/usr`.

```
CHGNFSEXP OPTIONS('-I -O ACCESS=Prof:1.234.5.6')
          DIR('/usr')
```

This command exports the user-defined file system (UDFS) that is named `kate.udfs` and makes it available to appropriate clients `Prof` and `1.234.5.6`.

For more information about the MOUNT and ADDMFS commands, see Chapter 5, "Client Mounting of File Systems" on page 43. For more information about the EXPORTFS and CHGNFSEXP commands, see Chapter 4, "Server Exporting of File Systems" on page 27.

Unmount a User-Defined File System

The Remove Mounted File System (RMVMFS) or UNMOUNT commands will make a mounted file system inaccessible to the integrated file system namespace. If any of the objects in the file system are in use (for example, a file is opened) at the time of using the unmount command, an error message will be returned to the user. If the user has mounted over the file system itself, then this file system cannot be unmounted until it is uncovered.

Note: Unmounting an exported UDFS which has been mounted by a client will cause the remote client to receive the `ESTALE` return code for a failed operation upon the next client attempt at an operation that reaches the server.

RMVMFS/UNMOUNT Display

For a display of the RMVMFS (Remove Mounted File System) and UNMOUNT commands, please see "RMVMFS (Remove Mounted File System) Command" on page 52.

For more information about the UNMOUNT and RMVMFS commands, see Chapter 5, "Client Mounting of File Systems" on page 43.

Saving and Restoring a User-Defined File System

The user has the ability to save and restore all UDFS objects, as well as their associated authorities. The Save command (SAV) allows a user to save objects in a UDFS while the Restore command (RST) allows a user to restore UDFS objects. Both commands will function whether the UDFS is mounted or unmounted.

Perform UDFS operations in iSeries Navigator

iSeries Navigator, a graphical user interface (GUI) for the iSeries server, provides easy and convenient access to user-defined file systems. iSeries Navigator allows you to create, delete, mount, and unmount a UDFS from a Windows** client.

To create a new user-defined file system (UDFS):

1. In **iSeries Navigator**, expand your server.
2. Expand **File Systems**.
3. Expand **Integrated File System**.
4. Expand **Root**.
5. Expand **Dev**.
6. Click the auxiliary storage pool (ASP) that you want to contain the new UDFS.
7. Select **New UDFS** from the **File** menu.
8. On the **New User-Defined File System** dialog, specify the UDFS name, description (optional), auditing values, default file format, and whether the files in the new UDFS will have case-sensitive file names.

The following graphics are examples of a UDFS in iSeries Navigator if you are connected to iSeries through your PC:

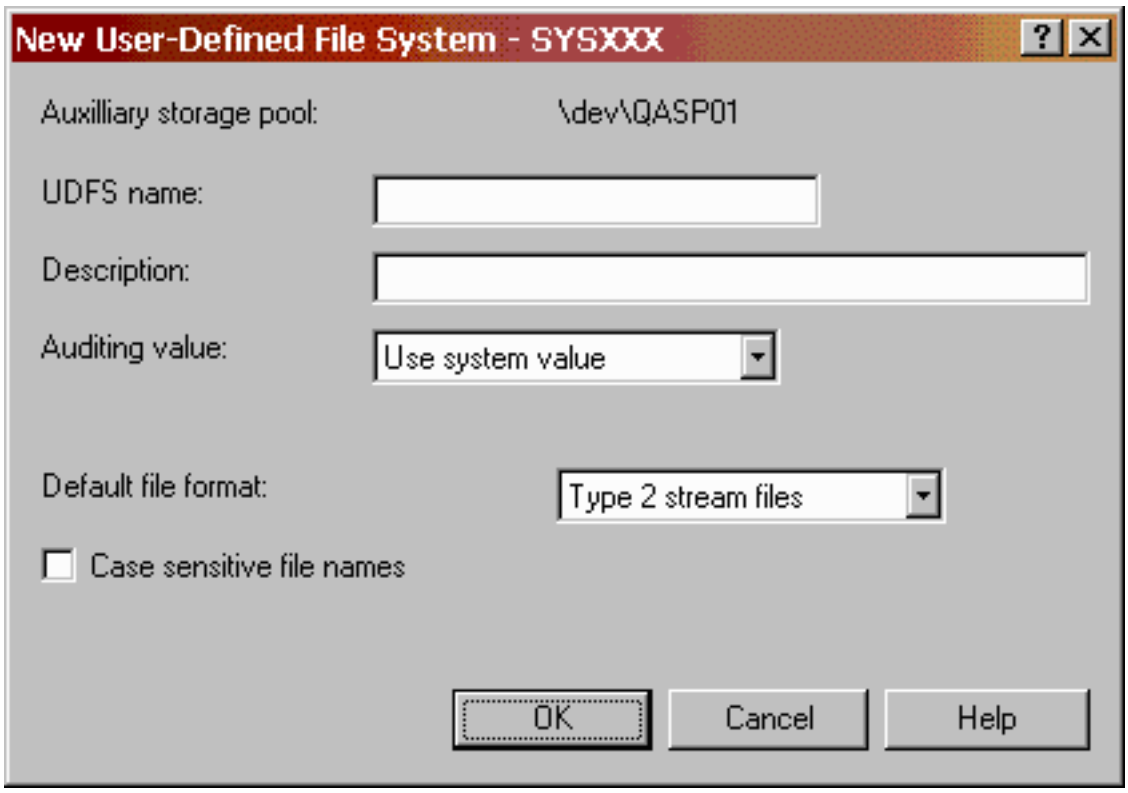

Figure 14. A Windows® view of using the CRTUDFS (Create UDFS) command

In iSeries Navigator, you can specify the name, auditing value, case-sensitivity, and other options of the UDFS. For more information about the CRTUDFS command, see "Create a User-Defined File System" on page 15.

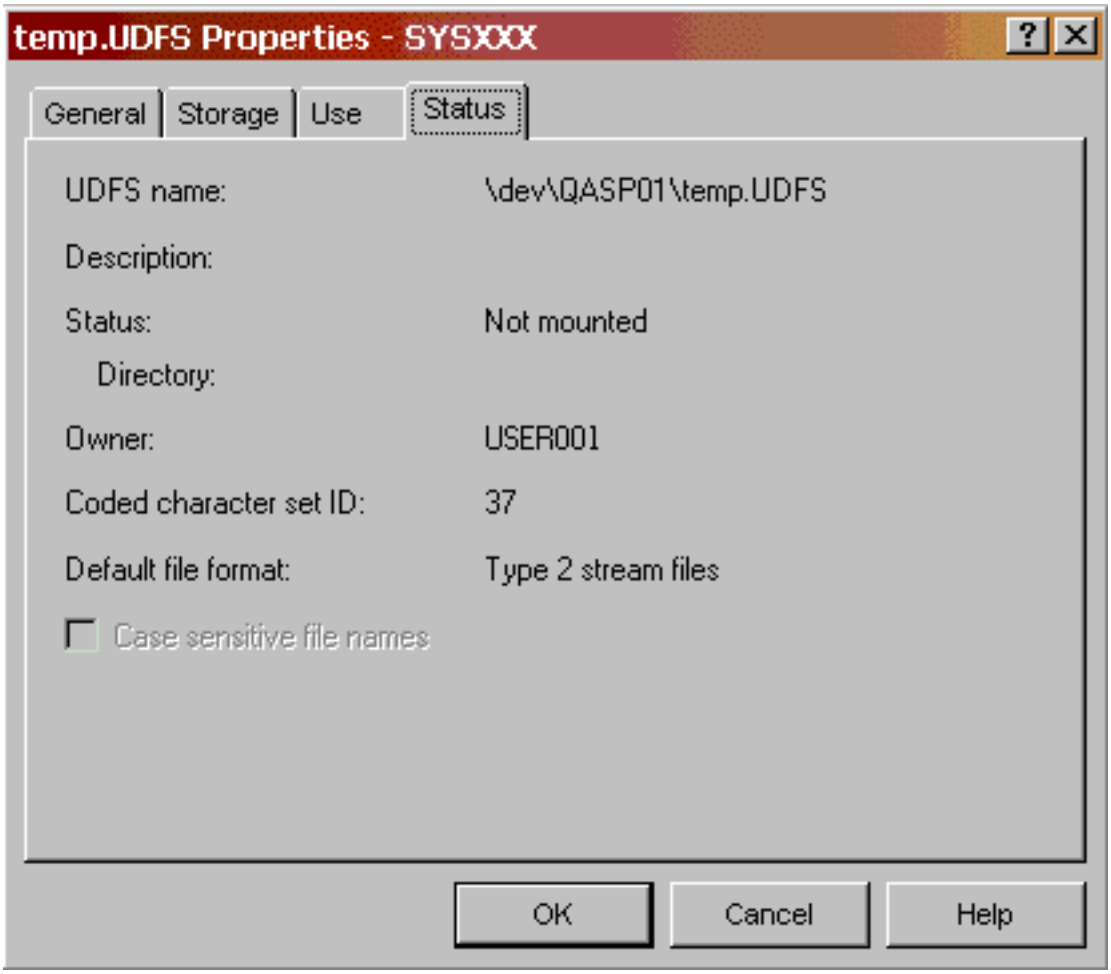

Figure 15. A Windows view of using the DSPUDFS (Display UDFS) command

In iSeries Navigator, you can display the properties of the UDFS. For more information about the DSPUDFS command, see "Display a User-Defined File System" on page 17.

Mount a UDFS in iSeries Navigator

When accessing an iSeries server using iSeries Navigator, you can dynamically mount user-defined file systems.

To mount a user-defined file system (UDFS):

1. In **iSeries Navigator**, expand your server.
2. Expand **File Systems**.
3. Expand **Integrated File System**.
4. Expand **Root**.
5. Expand **Dev**.
6. Click the auxiliary storage pool (ASP) that contains the UDFS that you want to mount.
7. In the **UDFS Name** column of iSeries Navigator's right pane, right-click the UDFS that you want to mount, and select **Mount**.

Tip: If you like to drag and drop, you can mount a UDFS by dragging it to a folder within the integrated file system on the same server. You cannot drop the UDFS on `/dev`, `/dev/QASPxx`, another server, or the desktop.

The following graphic shows the iSeries Navigator dialog box for mounting a user-defined file system:

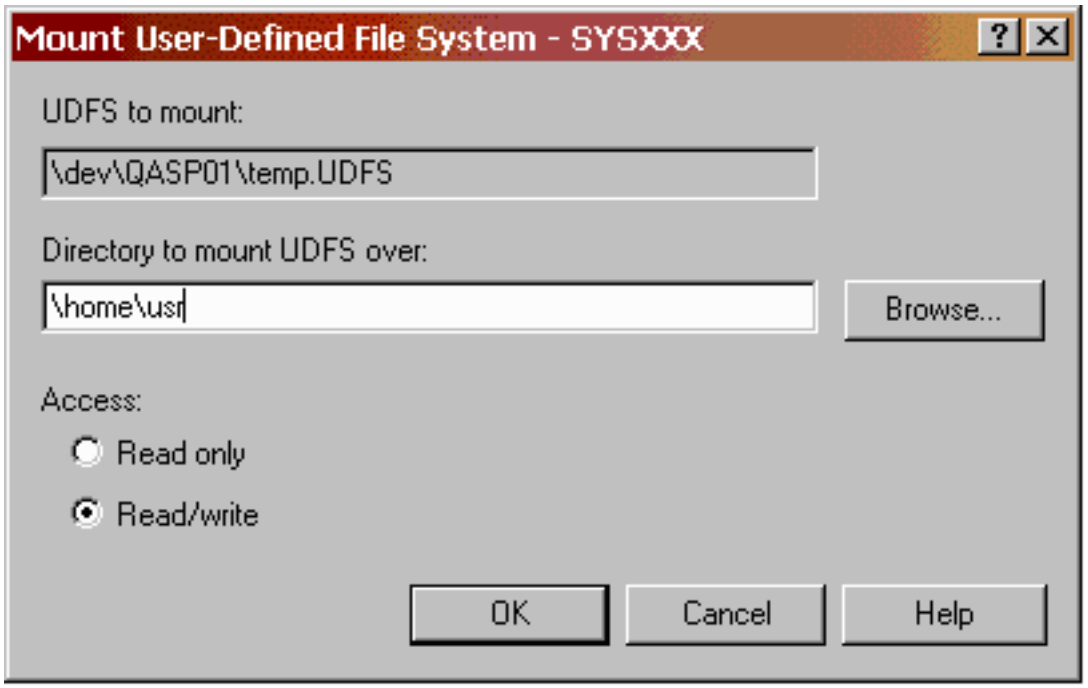

Figure 16. A Windows view of Mounting a user-defined file system

User-Defined File System Functions in the Network File System

To export the contents of a UDFS, you must first mount it on the local namespace. Once the Block Special File (*BLKFS) mounts, it behaves like the "root" or QOpenSys file systems. The UDFS contents become visible to remote clients when the server exports them.

It is possible to export an unmounted UDFS (*BLKSF object) or the ASP in which it resides. However, the use of such objects is limited from remote NFS clients. You will have minimal use of mounting and viewing from most UNIX clients. You cannot mount a *BLKSF object on iSeries clients or work with them in NFS mounted directories. For this reason, exporting /DEV or objects within it can cause administrative difficulties. The next sections describe how you can work around one such scenario.

Using User-Defined File Systems with Auxiliary Storage Pools

This scenario involves an eager user, a non-communicative system administrator, and a solution to an ASP problem through the Network File System.

A user, Jeff, accesses and works with the TULAB2 namespace each time he logs into his account on a remote NFS client. In this namespace exist a number of user-defined file systems (`A.udfs`, `B.udfs`, `C.udfs`, and `D.udfs`) in an ASP connected to the namespace as `/DEV/QASP02/`. Jeff is used to working with these directories in their familiar form every day.

One day, the system administrator deletes the UDFSs and physically removes the ASP02 from the server. The next time Jeff logs in, he can't find his UDFSs. So, being a helpful user, Jeff creates a /DEV/QASP02/ directory using the CRTDIR or MKDIR command and fills the sub-directories with replicas of what he had before. Jeff replaces A.udfs, B.udfs, C.udfs, and D.udfs with 1.udfs, 2.udfs, 3.udfs, and 4.udfs.

This is a problem for the network because it presents a false impression to the user and a liability on the server. Because the real ASP directories (/DEV/QASPXX) are only created during IPL by the system, Jeff's new directories do not substitute for actual ASPs. Also, because they are not real ASP directories (/DEV/QASPXX), all of Jeff's new entries take up disk space and other resources on the system ASP, not the QASP02, as Jeff believes.

Furthermore, Jeff's objects are not UDFSs and may have different properties than he expects. For example, he cannot use the CRTUDFS command in his false /DEV/QASP02 directory.

The system administrator then spontaneously decides to add a true ASP without shutting the system down for IPL. At the next IPL, the new ASP will be mounted over Jeff's false /dev/qasp02 directory. Jeff and many other users will panic because they suddenly cannot access their directories, which are "covered up" by the system-performed mount. This new ASP cannot be unmounted using either the RMVMFS or UNMOUNT commands. For Jeff and other users at the server, there is no way to access their directories and objects in the false ASP directory (such as 1.udfs, 2.udfs, 3.udfs, and 4.udfs)

Recovery with the Network File System

The NFS protocol does not cross mount points. This concept is key to understanding the solution to the problem described above. While the users at the server cannot see the false ASP and false UDFSs covered by the system-performed mount, these objects still exist and can be accessed by remote clients using NFS. The recovery process involves action taken at both the client and the server:

1. The administrator can export a directory above the false ASP (and everything "downstream" of it) with the EXPORTFS command. Exporting /DEV exports the underlying false ASP directory, but not the true ASP directory that is mounted over the false ASP directory. Because NFS does not cross the mount point, NFS recognizes only the underlying directories and objects.

   ```
   EXPORTFS OPTIONS('-I -O ROOT=TUclient52X') DIR('/DEV')
   ```

2. Now the client can mount the exported directory and place it over a convenient directory on the client, like /tmp.

   ```
   MOUNT TYPE(*NFS) MFS('TULAB2:/DEV') MNTOVRDIR('/tmp')
   ```

3. If the client uses the WRKLNK command on the mounted file system, then the client can now access the false ASP directory and connecting directories will be maintained.

   ```
   WRKLNK '/tmp/*'
   ```

4. The server then needs to export a convenient directory, like /safe, which will serve as the permanent location of the false ASP directory and its contents.

   ```
   EXPORTFS OPTIONS ('-I -O ROOT=TUclient52X') DIR('/safe')
   ```

5. The client can mount the directory /safe from the server to provide a final storage location for the false ASP directory and its contents.

   ```
   MOUNT TYPE(*NFS) DIR('/safe') MNTOVRDIR('/user')
   ```

6. Finally, the client can copy the false ASP directory and the false objects `1.udfs`, `2.udfs`, `3.udfs`, and `4.udfs` on the server by copying them to the `/safe` directory that has been mounted on the client.

   ```
   COPY OBJ('/temp/*') TODIR('/user')
   ```

The false QASP02 directory and the false objects that were created with it are now accessible to users at the server. The objects are now, however, located in `/safe` on the server.

Chapter 4. Server Exporting of File Systems

A key feature of the Network File System is its ability to make various local file systems, directories, and objects available to remote clients through the **export** command. Exporting is the first major step in setting up a "transparent" relationship between client and server.

Before exporting from the server, remote clients cannot "see" or access a given file system on the local server. Furthermore, remote clients are completely unaware of the existence file systems on the server. Clients cannot mount or work with server file systems in any way. After exporting, the clients authorized by the server will be able to mount and then work with server file systems. Exported and mounted file systems will perform as if they were located on the local workstation. Exporting allows the NFS server administrator a great range of control as to exactly what file systems are accessible and which clients can access them.

What is Exporting?

Exporting is the process by which users make local server file systems accessible to remote clients. Assuming that remote clients have the proper authorities and access identifications, they can see as well as access exported server file systems.

Using either the CHGNFSEXP or EXPORTFS command, you can add directory names from the `/etc/exports` file to the export table for export to remote clients. You can also use these commands to dynamically export from the NFS server, bypassing the `/etc/exports` file.

A host system becomes an NFS server if it has file systems to export across the network. A server does not advertise these file systems to all network systems. Rather, it keeps a list of options for exported file systems and associated access authorities and restrictions in a file, `/etc/exports`. The `/etc/exports` file is built into the **export table** by the export command. The command reads the export options and applies them to the file systems to be exported at the time the command is used. Another way of exporting file systems is to do so individually with the "-I" option of the export command. This command will not process any information stored in the `/etc/exports` file.

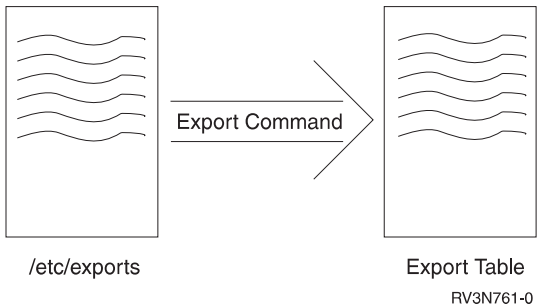

Figure 17. Exporting file systems with the /etc/exports file

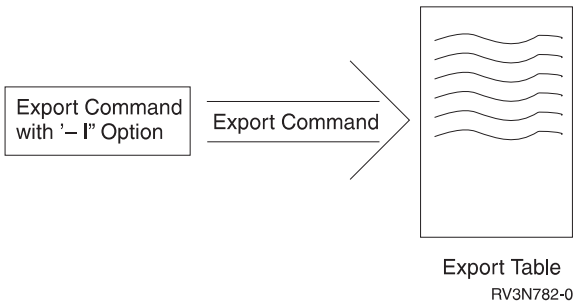

Figure 18. Dynamically exporting file systems with the "-I" option

The mount daemon checks the export table each time a client makes a request to mount an exported file system. Users with the proper authority can update the `/etc/exports` file to export file systems at will by adding, deleting, or changing entries. Then the user can use the export command to update the export table. Most system administrators configure their NFS server so that, as it starts up, it checks for the existence of `/etc/exports`, which it immediately processes. Administrators can accomplish this by specifying *ALL on the STRNFSSVR (Start Network File System Server) command. Once the server finds `/etc/exports` in this way, it uses the export command to create the export table. This makes the file systems immediately available for client use.

Why Should I Export?

Exporting gives a system administrator the opportunity to easily make any file system accessible to clients. The administrator can perform an export at will to fulfill the needs of any particular user or group of users, specifying to whom the file system is available and how.

With an efficient system of exporting, a group of client systems needs only one set of configuration and startup files, one set of archives, and one set of applications. The export command can make all of these types of data accessible to the clients at any time.

Although there are many insecure ways to export file systems, using the options on the export command allow administrators to export file systems safely. Exported file systems can be limited to a group of systems in a trusted community of a network namespace.

Using the ability to export allows for a simpler and more effective administration of a namespace, from setting up clients to determining what authority is needed to access a sensitive data set. A properly used `/etc/exports` file can make your namespace safe and secure while providing for the needs of all your users.

TULAB Scenario

In TULAB, a group of engineering undergraduate students are working with a group of engineering graduate students. Both sets of students have access to their remote home directories and applications on the server through NFS. Their research involves the controversial history of local bridge architecture. The students will be working in different rooms of the same campus building. Chris Admin needs a way to make data available to both groups of computers and students without making all the data available to everyone.

Chris Admin can export a directory containing only the database files with statistics of the bridge construction safety records. This operation can be performed without fear of unknown users accessing the sensitive data. Chris Admin can use the export command to allow only selected client systems to have access to the files. This way, both groups of students will be able to mount and access the same data and work with it on their separate, different workstations.

What File Systems Can I Export?

You can export any of the following file systems:
- "Root" (/)
- QOpenSys
- QSYS.LIB
- QDLS
- QOPT
- UDFS

You cannot export other file systems that have been mounted from remote servers. This includes entire directory trees as well as single files. iSeries servers follow the general rules for exporting as detailed in "Rules for Exporting File Systems" on page 30. This set of rules also includes the general NFS "downstream" rule for exporting file systems. Remember that whenever you export a file system, you also export all the directories and objects that are located hierarchically beneath ("downstream" from) that file system.

The CHGNFSEXP or EXPORTFS command is the key for making selected portions of the local user view part of the remote client view.

Exported file systems will be listed in the export table built from the `/etc/exports` file. These file systems will be accessible to client mounting, assuming the client has proper access authorities. Users will be able to mount and access exported data as if it were local to their client systems. The administrator of the server can also change the way file systems are exported dynamically to present remote clients with a different view of file systems.
Before exporting, a remote client cannot view any local server file systems.

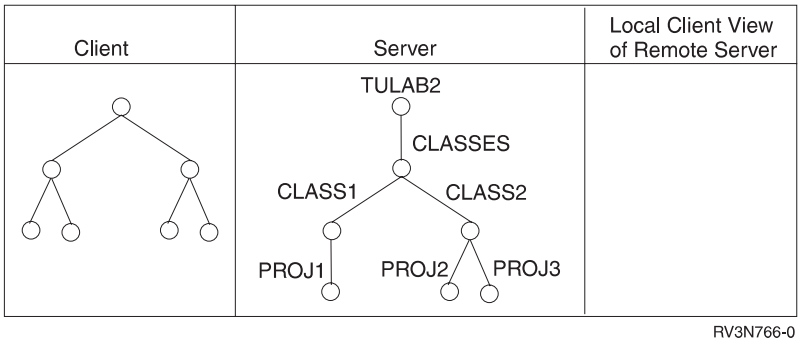

Figure 19. Before the server has exported information

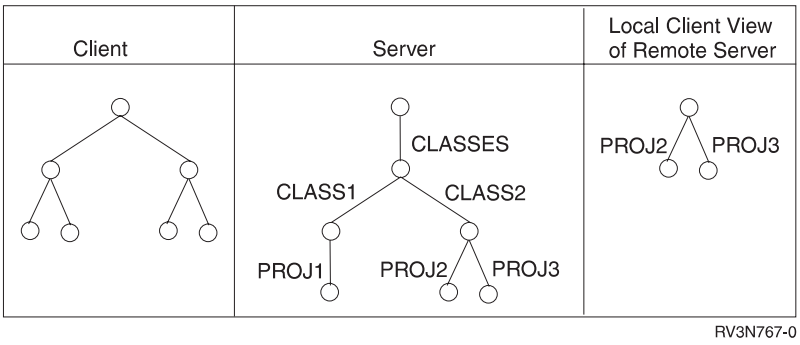

Figure 20. After the server has exported /classes/class2

After exporting, a remote client can view the exported file systems `PROJ2` and `PROJ3`.

Not all the file systems on the server are visible to remote clients. Only the exported file systems are available for mounting by clients with proper authorities as specified on the export command or in the `/etc/exports` file. Remote clients can not see anything except for their own local files and those that the various remote servers have exported. Before remote clients can access server data, that data must first be exported and then mounted.

How Do I Export File Systems?

You can export NFS server file systems for mounting on a client with the CHGNFSEXP (Change Network File System Export) or the EXPORTFS CL commands.

For a discussion of specific special considerations regarding export procedures, please see Chapter 9, “Network File System Security Considerations” on page 83.

Rules for Exporting File Systems

There are four conceptual rules that apply when exporting the file systems of an NFS server so that they are accessible to clients:

1. Any file system, or proper subset of a file system, can only be exported from a system that runs NFS. A proper subset of a file system is defined as a file or directory that starts below the path of the file system being exported. This capability allows you to export only certain parts of large file systems at any one time instead of the entire file system all at once.

   For example, `/usr` is a file system, and the `/usr/public_html` directory is part of that file system. Therefore, `/usr/public_html` (and all of the objects, files, and sub-directories located within that directory) is a proper subset of the `/usr` file system.

   You might want to think of the exporting process as you would think of a river. When you throw an object into a river, it flows downstream. When you export a file system, you also export all of the “downstream” directories, file systems, and other objects. Just as rivers flow from upstream to downstream, so do your exports.

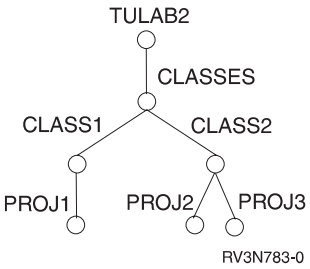

Figure 21. A directory tree before exporting on TULAB2

On any given server file system, when you start exporting, all the objects beneath the export point will also be exported. This includes directories, files, and objects.

For example, if you export `/classes` from TULAB2, then everything below

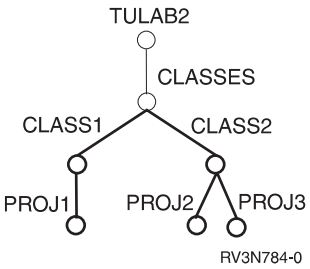

Figure 22. The exported directory branch /classes on TULAB2

`/classes` is also exported, including `/classes/class1`, `/classes/class2` and their associated sub-directories. If your clients only need access to `/classes/class1`, then you have exported too much data.

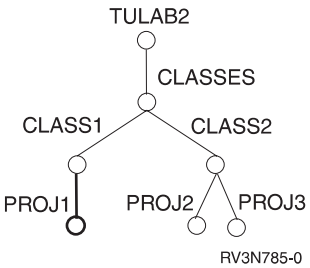

Figure 23. The exported directory branch /classes/class1 on TULAB2

A wiser decision is to only export what clients need. If clients need access only to `/classes/class1`, then export only `/classes/class1`. This makes work on the client easier by creating less waste on the client namespace. Furthermore, it guards against additional security concerns on the NFS server.

2. You cannot export any sub-directory of an already-exported file system unless the sub-directory exists on a different local file system.

   For example, the directory `/bin` has been exported, and `/bin/there` is a sub-directory of `/bin`. You cannot now export `/bin/there` unless it exists on a different local file system.

3. You cannot export any parent directory of an exported file system unless the parent is on a different local file system.

For example, the file system /home/sweet/home has been exported, and /home/sweet is a parent directory of /home/sweet/home. You cannot now export /home/sweet unless it exists on a different local file system.

4. You can only export local file systems. Any file systems or proper subsets of file systems that exist on remote servers cannot be exported except by those remote servers.

   For example, /help exists on a different server than the one you are currently accessing. You must be able to access that remote server in order to export /help.

   A more complicated example involves trying to export a file system which a local client mounts from a remote server. For example, the file system /home/troupe resides on a local client, and the file system /remote1 exists on a remote server. If the client mounts /remote1 over /home/troupe, then the client cannot export /home/troupe. This is because it actually exists on a remote server and not the local client.

The first rule allows you to export only selected portions of a large file system. You can export and mount a single file, a feature which is used extensively by clients without local disks. The second and third rules say that you can export a local file system in one way, and one way only. Once you export a sub-directory of a file system, you cannot go *upstream* and export the whole file system. Also, once you have made the entire file system public, you cannot restrict the *downstream* flow of the export to include only a few files.

Exporting sub-directories is similar to creating views on a relational database. You choose the portions of the database that a user needs to see, hiding information that is either extraneous or confidential. In this way, system administrators can limit access to sensitive material.

CHGNFSEXP (Change Network File System Export) Command

Purpose

The Change Network File System Export (CHGNFSEXP) command adds directory names to (*exports*) the list of directory trees that are currently exported to NFS clients (the export table). This command also removes (*unexports*) the list of directory trees that are currently exported to NFS clients. The flags in the OPTIONS list indicate what actions you want the CHGNFSEXP command to perform. see the CL topic in the iSeries Information Center for a complete description of CHGNFSEXP options.

A list of directories and options for exporting a file system and its contents is stored in the /etc/exports file. The CHGNFSEXP command allows you to export all of the directory trees specified in the file using the '-A' flag. CHGNFSEXP also allows you to export a single directory tree by specifying the directory name. When the directory tree being exported exists in the /etc/exports file, you can export it with the options specified there, or you can use the '-I' flag to override the options, specifying the new options on the CHGNFSEXP command.

You can also export a directory tree not previously defined in the /etc/exports file by providing the '-I' and the options for it on the CHGNFSEXP command. You can unexport directory trees by using the '-U' flag on the CHGNFSEXP command. You can unexport any file systems that you have previously exported, even if remote clients have mounted them. The result is that the NFS server will send the remote clients the ESTALE error number on their next attempt to operate on an object in that file system.

You can export to specific groups of clients by using the /etc/netgroup file. This file contains an alias for a group of clients to whom file systems will be exported. No other systems outside of the netgroup will be able to access the file systems. For more information about the /etc/netgroup file, see "/etc/netgroup File" on page 98.

Users can call this command by using the following alternative command name:

- EXPORTFS

For more information on how to edit the /etc/exports file, see "Edit stream files using the Edit File (EDTF) command" on page 95.

Restrictions:

The following restrictions apply when changing export entries:

1. The user must have *IOSYSCFG special authority to use this command.
2. The user must have read (*R) and write (*W) data authority to the /etc directory.

For more information about the CHGNFSEXP and EXPORTFS commands and the associated parameters, see the CL topic in the iSeries Information Center.

CHGNFSEXP/EXPORTFS Display

```
Change NFS Export (CHGNFSEXP)

 Type choices, press Enter.

 NFS export options . . . . . > '-I -O RO,ANON=199,ACCESS=Prof:1.234.5.6'




 Directory  . . . . . > '/engdata/mech'

                          Additional Parameters
 Host options:
   Host name  . . . . . .   TULAB1   Character value, *DFT
   Data file CCSID  . . . . .   850      1-65533, *BINARY, *ASCII...
   Path name CCSID . .  . . . .   850      1-65533, *ASCII, *JOBCCSID
   Force synchronous write  . . .   *SYNC    *SYNC, *ASYNC
               + for more values

                                                                      Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 24. Using the Change NFS Export (CHGNFSEXP) display

When you use the CHGNFSEXP or EXPORTFS commands, you can specify many parameters and options:

- The export options list contains some flags followed optionally by a list containing a character string of characteristics for the directory tree being exported. Each flag consists of a minus '-' followed by a character. The options you list here will appear in the OPTIONS parameter on the CHGNFSEXP command. The flags are separated by spaces. Only certain combinations of flags are allowed. If a combination that is not valid is detected, an error will be returned.

- The directory entry is the name of the directory that you want to export. The pathname you specify will be listed in the DIR parameter on the CHGNFSEXP command. This entry specifies the path name of the existing directory to be exported (made available to NFS clients) or unexported (made unavailable to NFS clients). This directory can not be a sub-directory or a parent of an already exported directory (unless it is in a different file system). This parameter is not allowed when the -A flag is specified on the OPTIONS parameter.
- The HOSTOPT parameter has four elements that specify additional information about the NFS clients that a directory tree is being exported to. If you do not specify a HOSTOPT parameter for a host name you are exporting the directory tree to, the defaults for each of the elements of the HOSTOPT parameter are assumed for that host.
  1. The name of the host for which you are specifying additional options. This host should be specified above in the `OPTIONS -O` list as a host that has access to the exported directory tree. Specify either a single host name that is an alias for an address of a single host or a netgroup name to be associated with these options. You can assign names to an internet address with the Work with TCP/IP host table entries option on the Configure TCP/IP menu (CFGTCP command). Also, a remote name server can be used to map remote system names to internet addresses.
  2. The network data file CCSID is used for data of the files sent and received from the specified HOST NAME (or netgroup name). For any hosts not specified on the HOSTOPT parameter, the default network CCSID (binary, no conversion) is used.

     When new files are created, they are tagged with this CCSID unless the value is *BINARY. If the file has a *BINARY value, then it is tagged with the network path name CCSID.
  3. The network path name CCSID is used for the path name components of the files sent to and received from the specified HOST NAME (or netgroup name). For any hosts not specified on the HOSTOPT parameter, the default network path name CCSID (ASCII) is used.
  4. The write mode specifies whether write requests are handled synchronously or asynchronously for this HOST NAME (or netgroup name). The default `*SYNC` value means that data may will be written to disk immediately. The `*ASYNC` value does not guarantee that data is written to disk immediately, and can be used to improve server performance.

     Note: The Network File System protocol has traditionally used synchronous writes. Synchronous writes do not return control of the server to the client until *after* the data has been written by the server. Asynchronous writes are not guaranteed to be written.

Examples

Example 1: Exporting all entries from `/etc/exports`.

```
CHGNFSEXP OPTIONS('-A')
CHGNFSEXP '-A'
```

Both of these commands export all entries that exist in the `/etc/exports` file.

Example 2: Exporting one directory with options.

```
CHGNFSEXP OPTIONS('-I -O RO,ANON=199,ACCESS=Prof:1.234.5.6')
          DIR('/engdata/mech')
          HOSTOPT((TULAB1 850 850))
```

This command exports the directory tree under the path name /engdata/mech as read-only. This command allows only two clients to mount this directory tree. It takes advantage of the positional parameters, which do not require keywords. It uses the HOSTOPT parameter to specify code pages for the host TULAB1.

Example 3: Exporting a directory to many netgroups.

```
CHGNFSEXP OPTIONS('-I -O ACCESS=students:LabSystems:Profs, ANON=-1')
          DIR('/home')
```

This command exports the directory tree under the path name /home to three netgroups defined in the /etc/netgroup file:

1. students
2. LabSystems
3. Profs

However, not all members of these netgroups have read/write access to the /home directory. Individual and group permissions still apply to the path. This command also specifies ANON=-1, which does not allow access to this system by anonymous users.

Example 4: Forcing read-only permissions on an export.

```
CHGNFSEXP OPTIONS('-I -O RO, RW=students:LabSystems, ANON=-1,
                   ACCESS=Profs:LabSystems:students')
          DIR('/classes')
```

This command exports the directory tree under the path name /classes to the netgroups Profs, LabSystems, and students. However, only students and LabSystems have read/write authority to the exported tree. This command also specifies ANON=-1, which does not allow access to this system by anonymous users.

Exporting from iSeries Navigator

You can also export local file systems using iSeries Navigator. Perform the following steps to export from iSeries Navigator:

1. In **iSeries Navigator**, expand your server.
2. Expand **File Systems**.

   The following figure shows an example of the iSeries Navigator interface:

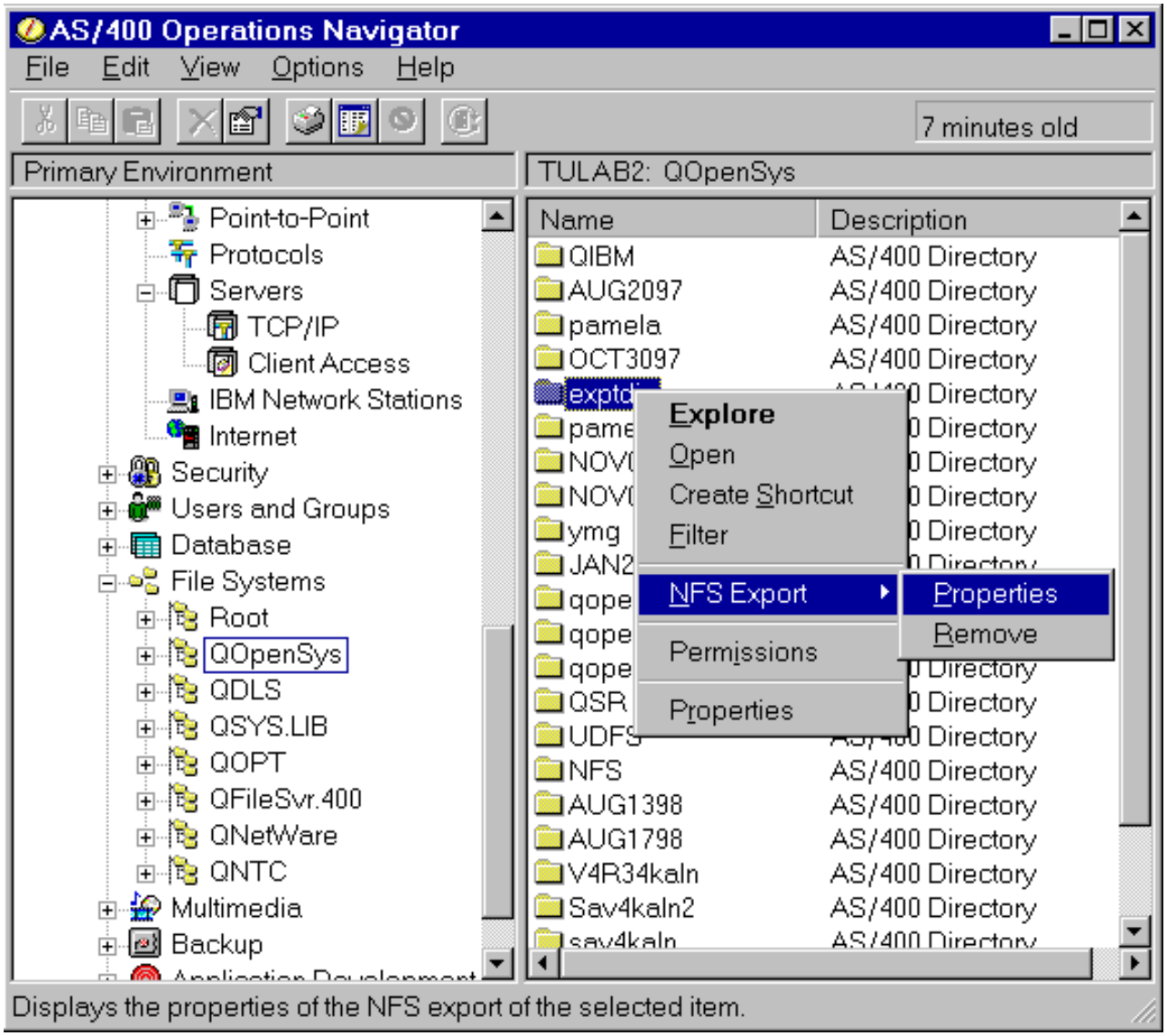

Figure 25. The iSeries Navigator interface.

3. Expand **Integrated File System**. Continue to expand until the file system or folder you want to export is visible.
4. Right click the file system or folder and select **NFS Export –> Properties**. The figure below shows the dialog box in iSeries Navigator:

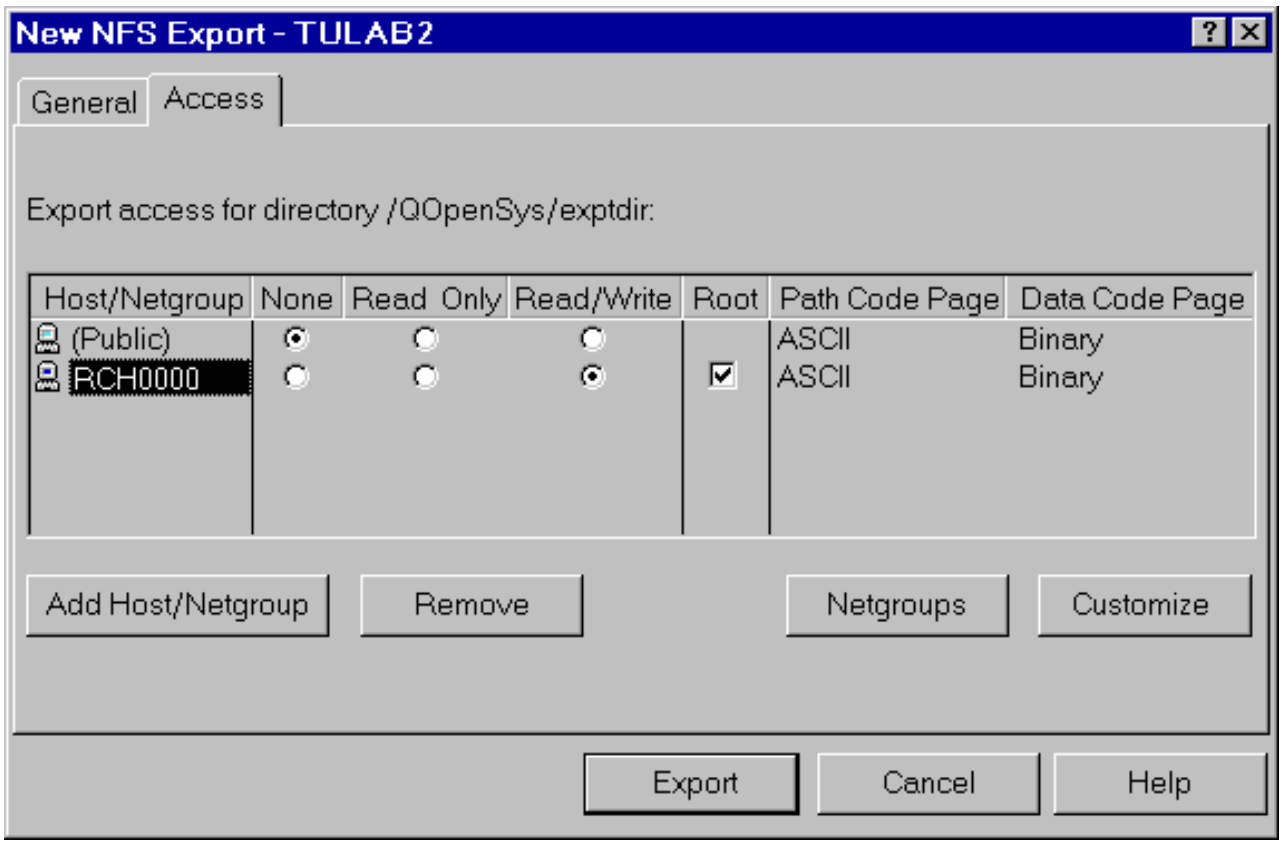

Figure 26. The NFS Export dialog box.

5. On the **General** tab, insert user ID that will be assigned to unknown or anonymous users who log on to the NFS server in the **Anonymous User Mapping** field.
6. Click the **Access** tab and select the appropriate access settings for the export.
7. Click **Add Host/Netgroup** to add other clients with specific privileges. The figure below shows a display of the Add Host/Netgroup dialog box:

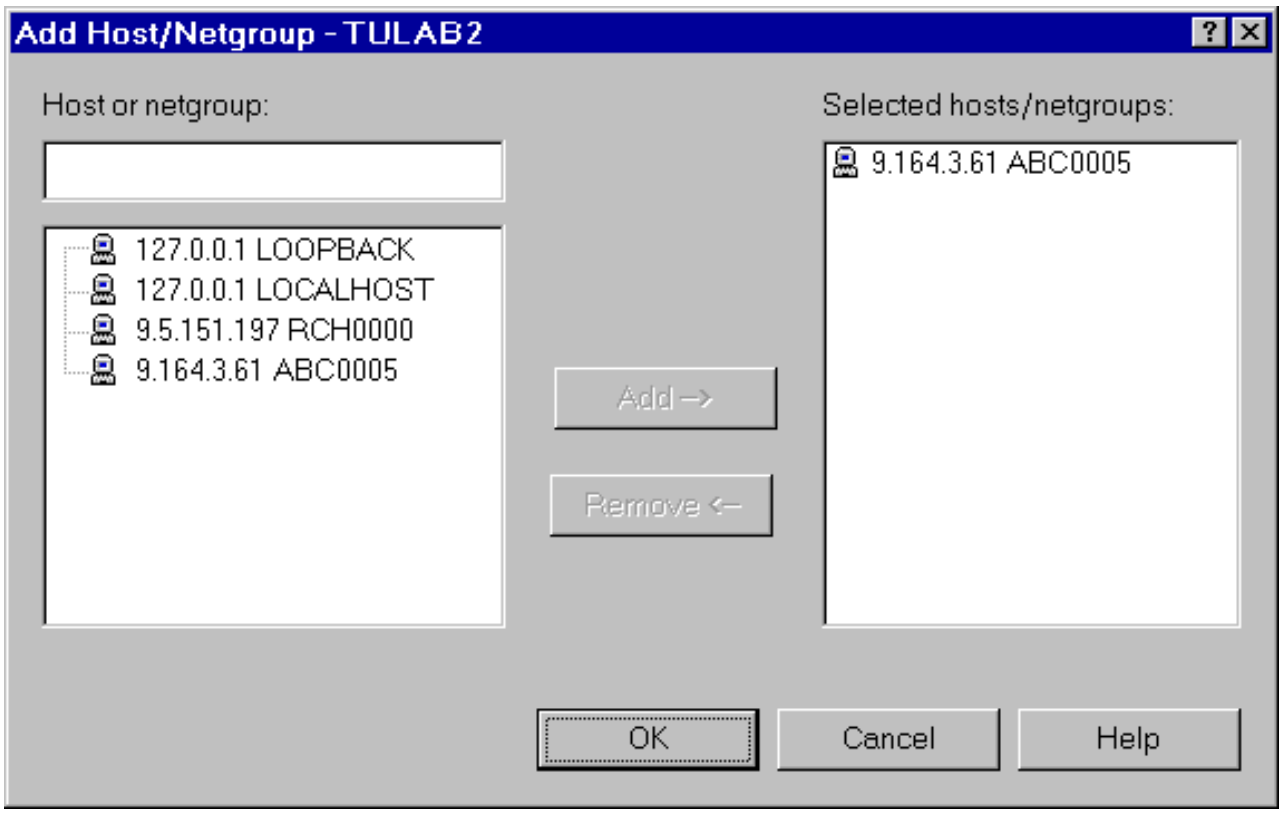

Figure 27. The Add Host/Netgroup dialog box.

8. Click **OK** to add your selected clients in the Add Host/Netgroup dialog box. You are brought back to the **New NFS Export** dialog.
9. Click **Customize** to configure the *Path CCSID* and *Data CCSID* options. The figure below shows an example of the **Customize NFS Clients Access** dialog box:

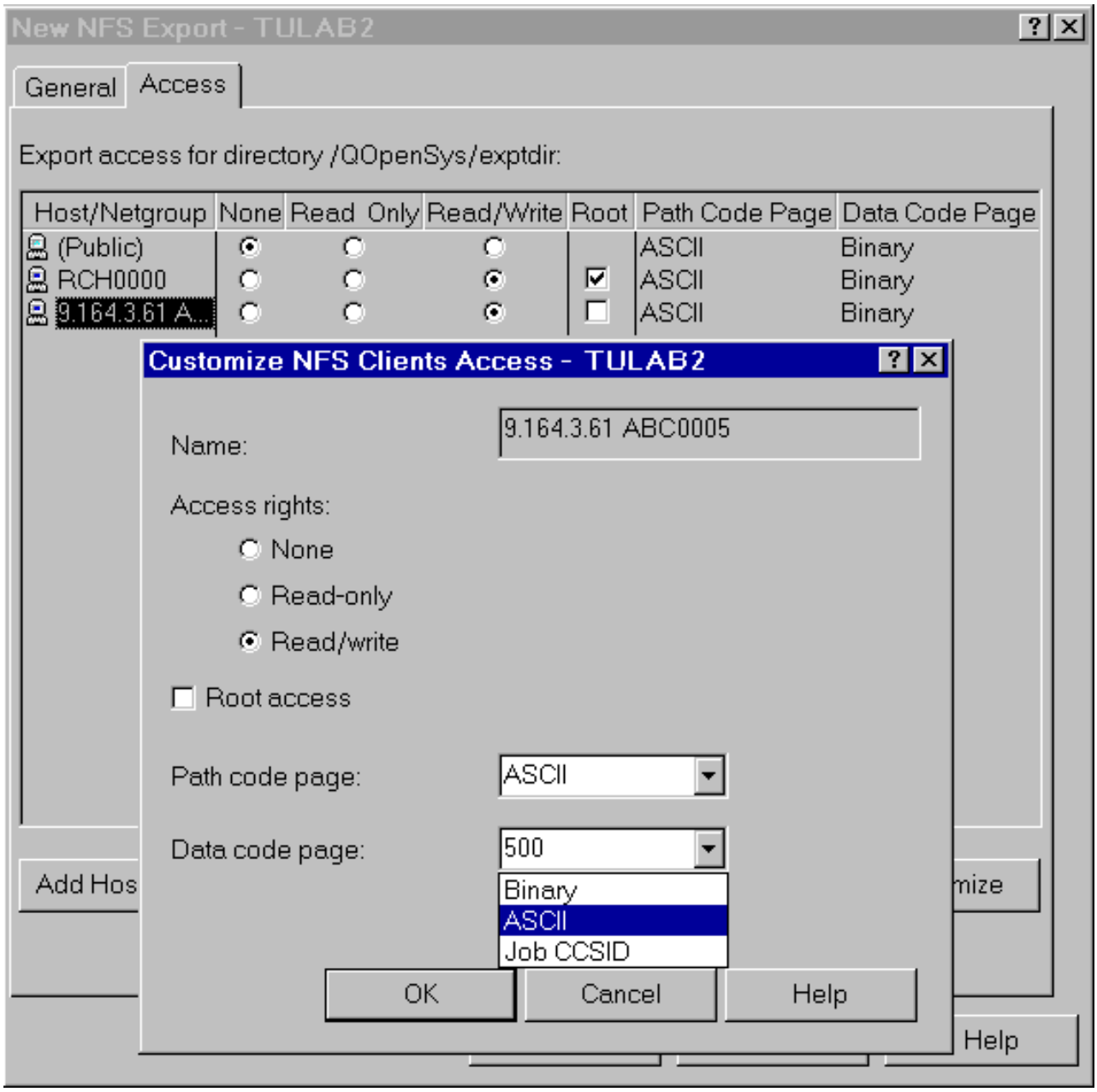

Figure 28. The Customize NFS Clients Access dialog box.

10. Click **OK** to add your customized settings to the export. You are brought back to the **New NFS Export** dialog.
11. Click **Export**.

    The new export will take place immediately. You also have the option of updating the `/ETC/EXPORTS` file with the new or changed export.

Find out what is exported

Often, you need to know the items that are currently exported on a server. There are three ways to do this:

1. Using iSeries Navigator
2. Using Retrieve Network File System Export Entries (QZNFRTVE) API

3. Using the UNIX `showmount` command on the network

Using iSeries Navigator

You can use iSeries Navigator to find objects that are currently exported on a server. To find objects that are currently exported:

1. In **iSeries Navigator**, expand your server.
2. Expand **Network**.
3. Expand **Servers**.
4. Click **TCP/IP**. The status of NFS and other servers are displayed in the right panel.
5. Right-click **NFS** and select **Exports**. From here, you can add new exports or remove entries from the list.

   The figure below shows the dialog box for NFS exports:

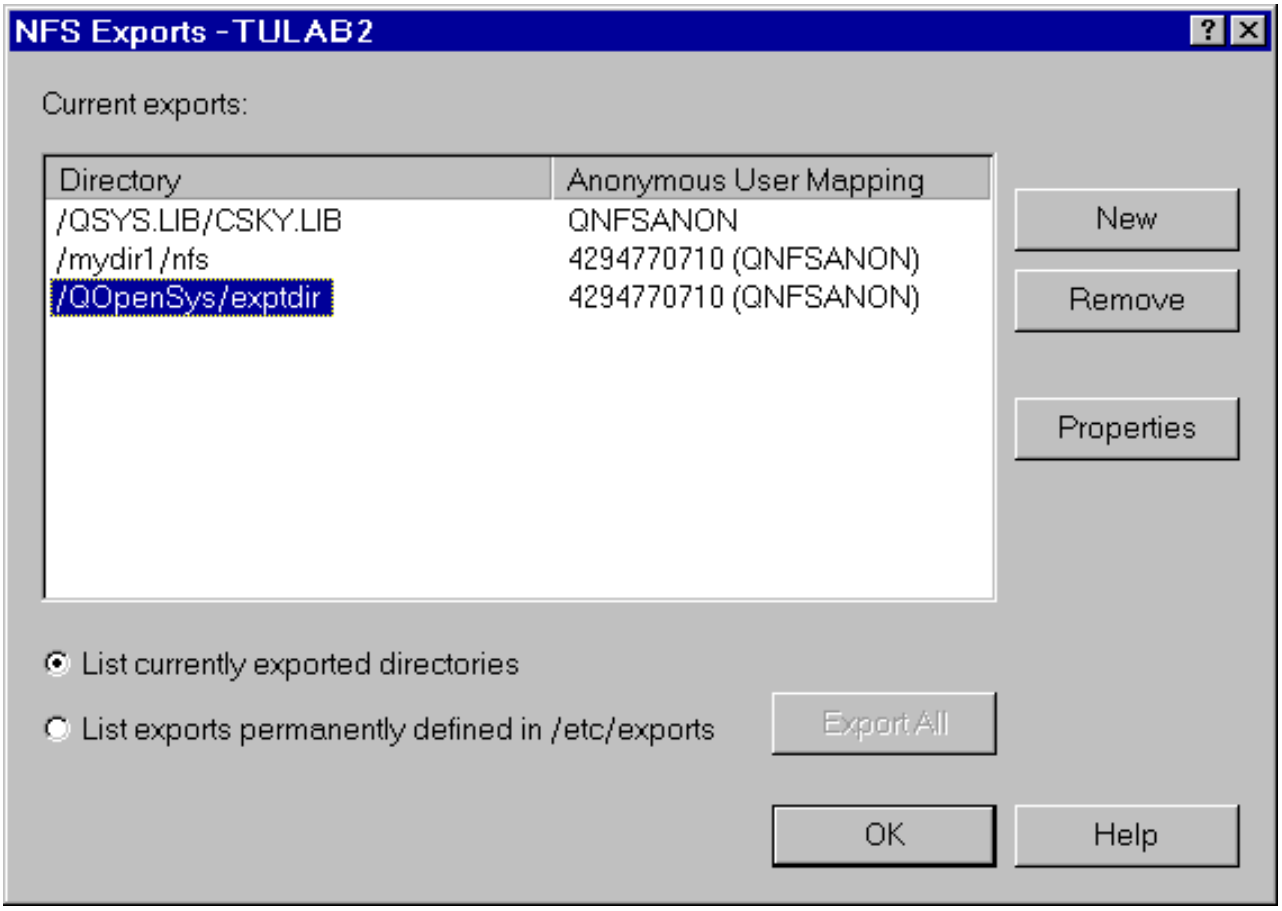

Figure 29. The NFS exports dialog box.

6. Click **OK**.

Retrieve Network File System Export Entries (QZNFRTVE) API

A second method of finding currently exported items on an iSeries is using the Retrieve Network File System Export Entries (QZNFRTVE) API. This programming interface allows you to retrieve all of the information about one or more exported entries. To call this API from the command line, type:

```
CALL QZNFRTVE
```

This lists all of the current exports if you are viewing detailed messages. Hitting PF1 on an individual export displays the configuration details for that entry. For more information about the QZNFRTVE API, please refer to the API topic in the iSeries Information Center.

UNIX showmount command

A third method involves displaying the exports from a UNIX system on the network. The Mount daemon provides the service of returning the exports. The following command issued on TULAB1 shows how to use this service to find the exported entries on TULAB2:

```
> showmount -e tulab2
export list for tulab2:
/classes/class1 host1,host2
/engdata/civil (everyone)
```

Exporting Considerations

Mounted File System Loops

Users and administrators can encounter difficulty with the inability of NFS to export an already-mounted Network File System. NFS will not allow the export of a mounted file system because of the possibility of mounted file system loops. This problem would occur if NFS allowed the mount and then new clients mounted the namespace. The new clients would then discover that to *find* the mount point, they would have to *pass through* the mount point. They would never get to the bottom of the directory, which would keep recycling the path to infinity. The mount daemon would run into a never-ending string, thus causing client failure.

Mounted File System Loops Solution

This problem will not occur because NFS will not allow the export of an already-mounted NFS file system.

Symbolic Links

The Network File System supports the use of **symbolic links.** Symbolic links are the representations of path names that are in the form of a path contained in a file. The actual path is determined by doing a path search based on the contents of the file.

Two path names can lead to the same object because one is a symbolic link. For example, the directory `/nancy` can contain a symbolic link to `/home/grad/kathryn` so that the path to an object in `/home/grad/kathryn` can also be `/nancy`. The two path names are the same. If a user exports both entries, then both will export normally. Exporting `/home/grad/kathryn` is the *same* as exporting `/nancy`. NFS exports *objects*, and not *path names*, so the last occurrence of the object export will be the only one that requires saving in the export table.

For more information about symbolic links, see Symbolic links in the **File systems and management** topic of the iSeries Information Center.

Chapter 5. Client Mounting of File Systems

The **mount** command places the remote file system over a local directory on an NFS client. After exporting, mounting a file system is the second major step in setting up a "transparent" relationship between client and server.

Mounting allows clients to actually make use of the various file systems that the server has exported. Clients can use the mount command to map an exported file system over all or just part of a local file system. This action occurs so seamlessly that local applications will probably not distinguish between file systems mounted from a remote server and file systems existing locally. Multiple clients can mount and work with a single or multiple file systems at the same time.

Once file systems have been exported from a remote server, clients can then mount these accessible file systems and make them a part of their local namespace. Clients can dynamically mount and unmount all or part of exported server file systems. Once a client has mounted a file system onto its own local namespace, any local file system information below the mount point will be "covered up." This renders the "covered" or "hidden" file system inaccessible until the remote file system is unmounted.

What Is Mounting?

Mounting is a client-side operation that gives the local client access to remote server file systems. The mount command does not copy the file systems over to the client. Rather, it makes the remote file systems appear as if they physically exist on the client. In reality, the file systems exist only on the server and the client is only accessing them. The interface, however, is designed to give the impression that the mounted file systems are local. In most cases, neither applications nor users can tell the difference.

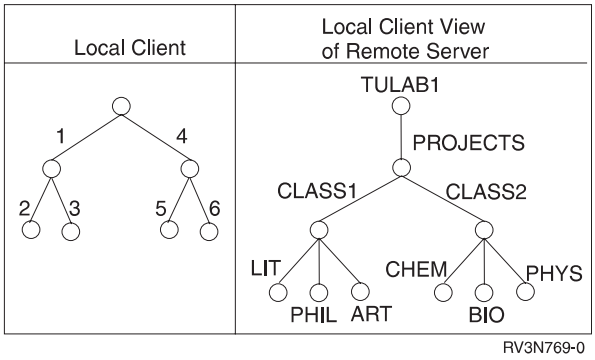

Figure 30. A local client and remote server with exported file systems

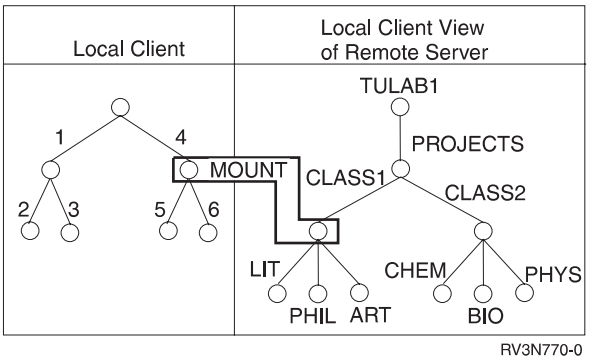

Figure 31. A local client mounting file systems from a remote server

Given the proper authority, an NFS client can mount any file system, or part of a

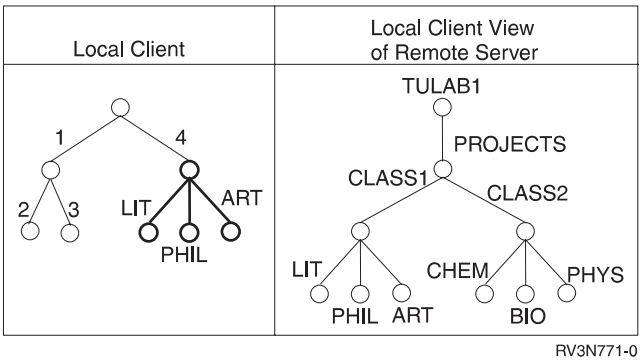

Figure 32. The mounted file systems cover local client directories

file system, that has been exported from an NFS server. Mounting is the local client action of selecting an exported directory from a remote server and making it accessible to the integrated file system namespace of the local client.

In many UNIX implementations of NFS, a user can list a remote file system in the `/etc/fstab` file on the client where it is automatically mounted after IPL. On iSeries NFS implementation, however, there is a program that operates at IPL that is typically used to start up various user-selectable system tasks. The name of this IPL-time startup program is stored in a system value called `QSTRUPPGM` The default name of the program is QSTRUP, which is located in the QSYS library. Users can edit this program to include the ADDMFS (Add Mounted File System) or MOUNT commands that will automatically mount remote file systems during startup. It is recommended that you use the STRNFSSVR *ALL command before using any mount commands. Remote exported file systems can also be explicitly mounted at any time after IPL by clients with proper authority using the ADDMFS or MOUNT commands.

When file systems are mounted on the client, they will “cover up” any file system, directories, or objects that exist beneath the mount point. This means that mounted file systems will also cause any file systems, directories, or objects that exist locally downstream from the mount point to become inaccessible.

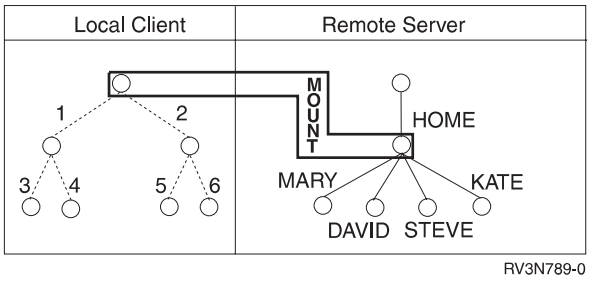

Figure 33. The local client mounts over a high-level directory

There is a “downstream” principle for mounting that is similar to the “downstream” rule for exporting. Whenever you mount a remote file system over a local directory, all of the objects “downstream” of the mount point are “covered up”. This renders them inaccessible to the local namespace. If you mount at a high level of a local directory tree, then you will cause most of your objects to become inaccessible.

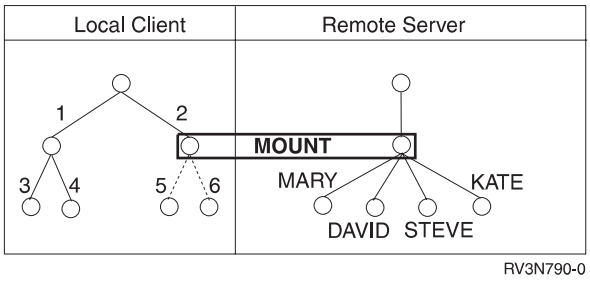

Figure 34. The local client mounts over the /2 directory

If you mount remote file systems over a lower point of the local directory tree, then you will not “cover up” as many of your objects, if any.

This “covering” aspect of mounting causes whatever local data beneath the newly mounted file system to become “invisible” to the user. They will not appear in the integrated file system namespace. “Covered” file systems are inaccessible until the mounted file system is unmounted. The client can dynamically **unmount** any file system that has previously been mounted. This action allows for the “uncovering” of a file system, directory, or object that has been mounted over. It also breaks the connection with the server (and therefore access) for that particular mounted file system.

Why Should I Mount File Systems?

Mounting file systems gives a client system the ability to work with files and other objects that were previously only available on the server. This is helpful in dispersing needed or important information to a wide variety of users and systems. Mounting also makes for less repetition when starting up a network. Each client can mount a startup configuration directly from the server that can be re-configured spontaneously, if necessary.

Mounting gives the remote client ease and freedom in deciding how to structure directories. File systems can be dynamically mounted and unmounted at will. Furthermore, users can specify parameters and options on the mount command that give clients and servers the most resources and highest security possible.

Sometimes the namespace of a client can become too complicated or overwhelmed with information. The unmount command is an easy way to slowly disengage from the server one file system at a time. To unmount all file systems, specify the *ALL value for the TYPE parameter on the UNMOUNT or RMVMFS (Remove Mounted File System) commands.

For detailed information on how to mount and unmount file systems, see "ADDMFS (Add Mounted File System) Command" on page 49 and "RMVMFS (Remove Mounted File System) Command" on page 52.

What File Systems Can I Mount?

Users can mount three different types of file systems on iSeries:

Network File Systems
Despite the fact that users will mount most file systems at startup time, there may be a need to dynamically mount and unmount file systems. Remote file systems exported by the server can be mounted at any time, assuming the local client has appropriate access authorities.

User-Defined File Systems
On iSeries, a UDFS is a local object that is visible as an opaque object in the integrated file system namespace. The contents of a UDFS are accessible only when it has been mounted within the integrated file system namespace. Although UDFSs can be mounted at the time of startup, users can dynamically mount and unmount a UDFS at any time.

Novell NetWare** file systems**
Users may also dynamically mount and unmount NetWare file systems. To learn more about NetWare file systems, see
- NetWare on iSeries in the iSeries Information Center
- NetWare file system in the iSeries Information Center

Where Can I Mount File Systems?

It is possible to mount an NFS file system over all or part of another client file system. This is possible because the directories used as mount points appear the same no matter where they actually reside.

To the client, NFS file systems appear to be and function as "normal," local file systems. Users can mount network file systems over the following iSeries client file systems:
- "Root" (though not over the root directory itself)
- QOpenSys
- NFS
- UDFS

When a client mounts an exported file system, the newly mounted file system will cover up whatever is beneath it. This is true for mounting remote file systems over local directories as well as mounting remote file systems over previously-mounted remote file systems. Any file system that is covered up in such a manner is inaccessible until all of the file systems "on top" are unmounted.

For example, TULAB2 exports /classes/class1, which contains the directory /classes/class1/proj1. A remote client has a local directory /user, which contains the directory /user/work, which contains the directory /user/work/time. The client mounts /classes/class1/ over /user/work, which causes the mounted file system

to completely cover up everything on the local directory tree that is “downstream” from the mount point. The mount point is `/user/work`. The `/user/work` directory now contains only `proj1`. Should the client try to access the data that is “covered up,” (`/user/work/time`) an error message returns to the user. Data that has been covered by a mounted file system is inaccessible until the file system is unmounted.

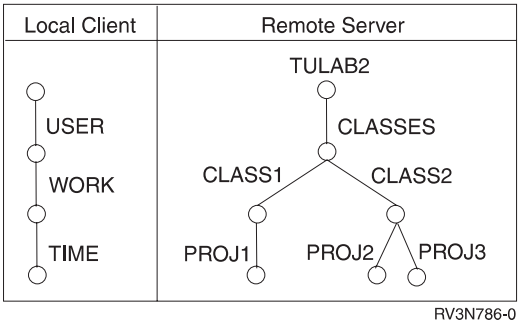

Figure 35. Views of the local client and remote server

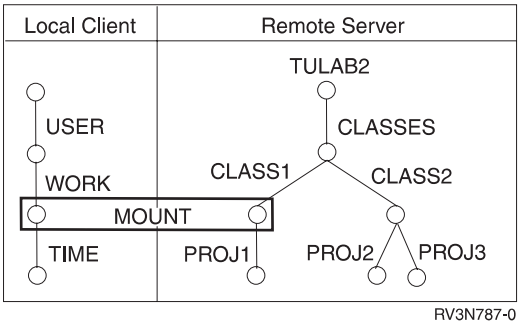

Figure 36. The client mounts /classes/class1 from TULAB2

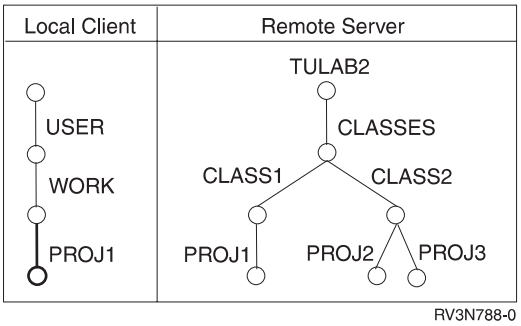

Figure 37. The /classes/class1 directory covers /user/work

To continue this example, another remote file system is exported by TULAB2, `/engdata`. In keeping with the “downstream” rule of exporting, all of the sub-directories of `/engdata` are also exported. The client can mount the exported sub-directory over the mount point that already exists. When a user mounts `/engdata` over the directory `/user/work`, all of the contents of `/user/work`, including `/user/work/proj1` become covered by the mount. This renders them inaccessible. The new local directory tree on the client will display `/user/work` and the various contents and sub-directories, as shown here.

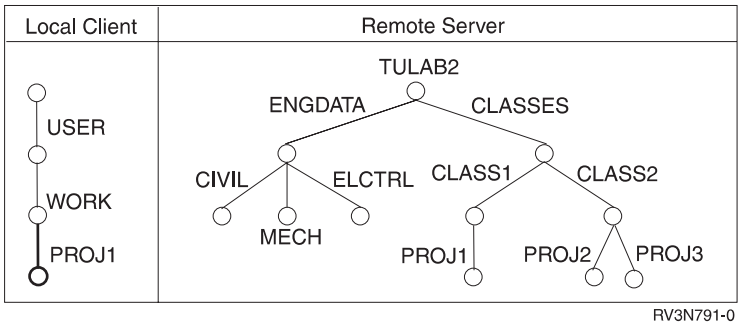

Figure 38. The remote server exports /engdata

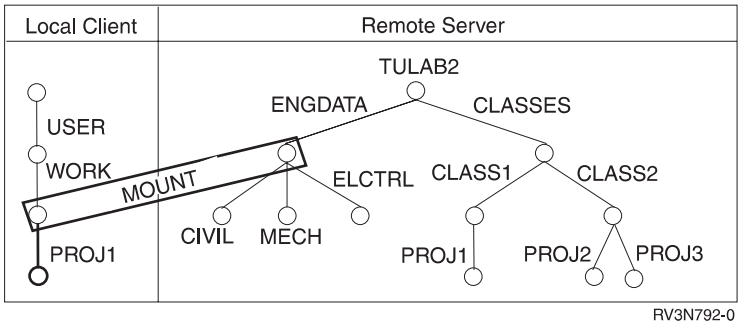

Figure 39. The local client mounts /engdata over a mount point

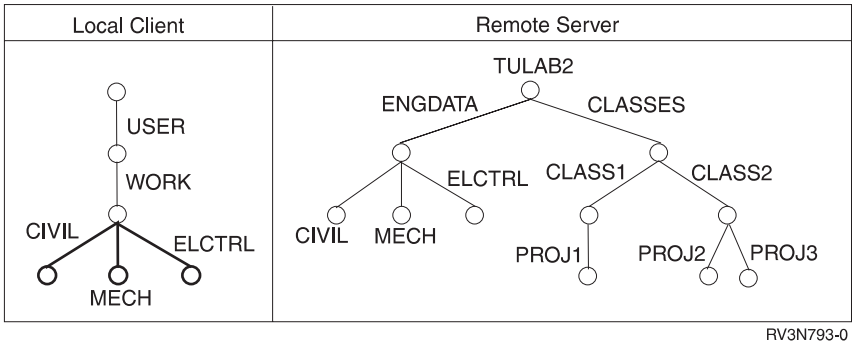

Figure 40. The /engdata directory covers /user/work

Note: NFS clients will *always* see the most recent view of a file system. If the client dynamically mounts or unmounts a file system, the change will be reflected on the namespace of the client after the next refresh.

It is possible to mount several different remote file systems over the same mount point. Users can mount file systems from various remote servers over the same directory on a local client without any prior unmounting. When unmounting the "stacked" file systems, the last mount is the first to be removed.

Mount Points

Mount points mark the area of the local client and remote server namespaces where users have mounted exported file systems. Mount points show where the file system has been *mounted from* on the server and show where it is *mounted to* on the client.

For example, the system exports the /home/consults directory from TULAB1 and mounts it over the /test directory on a remote client. The mount point on the client is /test. The old directory, **/test**, and its contents are covered up, and it becomes a window into the namespace of the server, TULAB1. To see which remote file system corresponds with a mount point or path, use the DSPMFSINF command. For example:

```
DSPMFSINF OBJ('/test')
```

For more information on this command, see "DSPMFSINF (Display Mounted File System Information) Command" on page 53.

How Do I Mount File Systems?

Users make remote server file systems accessible to the local namespace using the MOUNT and ADDMFS (Add Mounted File System) CL commands. The UNMOUNT and RMVMFS (Remove Mounted File System) commands will remove a mounted file system from the namespace. The DSPMFSINF (Display Mounted File System Information) command will provide information about a mounted file system. You can also reach these commands through a menu. Type GO CMDMFS (Go to the Mounted File System Commands menu) at any command line.

Before attempting to mount NFS file systems, you need to verify that the correct TCP/IP configuration exists on your server. Refer to the Configure TCP/IP topic in the iSeries Information Center for more information.

ADDMFS (Add Mounted File System) Command

Purpose

The Add Mounted File System (ADDMFS) command makes the objects in a file system accessible to the integrated file system name space. The file system to be mounted can be either:

1. A user defined file system (*UDFS) on the local server
2. A remote file system accessed via a local Network File System client (*NFS)
3. A local or remote NetWare file system (*NETWARE)

The directory that is the destination for the mount must exist. After completion of the mount, the contents of this directory will be "covered up" and rendered inaccessible to the integrated file system namespace.

Users can issue this command by using the following alternative command name:

- MOUNT

Restrictions:

1. You must have *IOSYSCFG special authority to use this command.
2. If you are mounting a user-defined file system or a Network File System, then you require *R (read) authority to the file system being mounted.
3. If you are mounting a NetWare file system, then you require *X (execute) authority to the file system being mounted.
4. You must have *W (write) authority to the directory being mounted over.

For more information about the ADDMFS and MOUNT commands and the associated parameters and options, see the CL topic in the iSeries Information Center.

ADDMFS/MOUNT Display

```
Add Mounted FS (ADDMFS)

 Type choices, press Enter.

 Type of file system  . . . . . > *NFS   *NFS, *UDFS, *NETWARE
 File system to mount . . . . . > 'TULAB2:/QSYS.LIB/SCHOOL.LIB'

 Directory to mount over  . . . . > '/HOME'
 Mount options  . . . . . .   'rw,suid,retry=5,rsize=8096,wsize=8096,timeo
=20,retrans=5,acregmin=30,acregmax=60,acdirmin=30,acdirmax=60,hard'


 Coded character set ID:
Data file CCSID  . . . . . .   *BINARY   1-65533, *ASCII, *JOBCCSID...
Path name CCSID  . . . . . .   *ASCII    1-65533, *ASCII, *JOBCCSID
 Code page:
   Data file code page  . . . . .   *BINARY   1-32767, *ASCII, *JOBCCSID...
   Path name code page  . . . . .   *ASCII    1-32767, *ASCII, *JOBCCSID


                                                                      Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel F13=How to use this display
 F24=More keys
```

Figure 41. Using the Add Mounted FS (ADDMFS) display

When you use the ADDMFS or MOUNT commands, you can specify many parameters and options:

- The required TYPE parameter on the ADDMFS command specifies the type of file system being mounted. The type of mount determines the correct form for the MFS parameter.
- The required MFS parameter on the ADDMFS command specifies the path name of the file system to be mounted. It can be the path to a local Block Special File (*BLKSF), a remote NFS path name, or the path of a NetWare file system.
- The required MNTOVRDIR parameter on the ADDMFS command specifies the path name of the existing directory that the file system will be mounted over. This directory gets "covered" by the mounted file system.
- The mount options list contains a character string of mount options. The keywords are separated by commas. For some keywords, an equal '=' and a value follow the keyword. If a keyword is not specified, the default value for that option will be used. The options list may contain spaces.
- The coded character set ID (CCSID) parameter on the ADDMFS command specifies a pair of CCSIDs for NFS. The CCSID values should be for pure single or double-byte CCSIDs.
- The data file CCSID specifies what CCSID should be assumed for data files on the remote server. You should specify a CCSID that has the same number of bytes per character as the original data.
- The path name CCSID specifies what CCSID should be assumed for path names on the remote server. This is a CCSID to be assumed for path names on the remote server.

Examples of the MOUNT command

Example 1: Mounting a User-Defined File System.

```
ADDMFS  TYPE(*UDFS) MFS('/DEV/QASP02/PROJ.UDFS')
        MNTOVRDIR('/REPORT')
```

This command mounts a user-defined file system PROJ.UDFS over the directory /report. This command uses the defaults for the other parameters.

Example 2: Mounting a Network File System from TULAB2.

```
ADDMFS  TYPE(*NFS) MFS('TULAB2:/QSYS.LIB/SCHOOL.LIB')
        MNTOVRDIR('/HOME')
```

This command mounts the remote /qsys.lib/school.lib file system from the remote system TULAB2 over the directory /home on a local client.

Example 3: Mounting a Network File System with Options.

```
ADDMFS  TYPE(*NFS) MFS('TULAB2:/QSYS.LIB/WORK.LIB')
        MNTOVRDIR('/HOME')
        OPTIONS('ro, nosuid, rsize=256, retrans=10')
        CCSID(*JOBCCSID)
```

This command mounts the /qsys.lib/work.lib file system from the remote system TULAB2 onto the local client directory /HOME. This command also specifies:

- Mount as read-only
- Disallow setuid execution
- Set the read buffer to 256 bytes
- Set the retransmission attempts to 10

The default job CCSID is used to determine the code page of the data on the remote system.

Example 4: Mounting a NetWare File System with Options.

```
ADDMFS  TYPE(*NETWARE)
        MFS('TULAB2/NET:WORK/PROJONE')
        MNTOVRDIR('/temp1')
        OPTIONS('ro,agregmax=120')
```

This command mounts the NetWare directory WORK/PROJONE contained in the volume NET that resides on server TULAB2 over the directory /temp1. In addition, this command specifies to mount as read-only, sets the maximum time to store file attributes locally to 120 seconds.

Example 5: Mounting using a NetWare Directory Services Context**.

Following are several examples of mounting a NetWare file system by using NetWare Directory Services (NDS**) contexts.

```
ADDMFS  TYPE(*NETWARE) MFS('.COMP.TULAB.UNIVER')
        MNTOVRDIR('/temp1')
```

This command mounts NDS volume COMP, using a distinguished context, over the directory /temp1.

```
ADDMFS  TYPE(*NETWARE)
        MFS('CN=NET_VOL.OU=TULAB2:WORK/PROJONE')
        MNTOVRDIR('/temp1')
```

This command mounts path WORK/PROJONE on NDS volume NET, using a relative path and fully qualified names, over the directory /temp1.

```
ADDMFS  TYPE(*NETWARE)
        MFS('.CN=NETMAP.OU=COMP.O=TULAB')
        MNTOVRDIR('/temp1')
```

This command mounts a directory map object, using a distinguished context and fully qualified names, over the directory `/temp1`.

RMVMFS (Remove Mounted File System) Command

Purpose

The Remove Mounted File System (RMVMFS) command will make a previously mounted file system inaccessible within the integrated file system name space. The file system to be made inaccessible can be:

1. a user defined file system (*UDFS) on the local server
2. a remote file system accessed via a Network File System server (*NFS)
3. a local or remote NetWare file system (*NETWARE).

If any of the objects in the file system are in use, the command will return an error message to the user. Note that if any part of the file system has **itself** been mounted over, then this file system cannot be unmounted until it is uncovered. If multiple file systems are mounted over the same mount point, the last to be mounted will be the first to be removed.

Users can also issue this command by using the following alternative command name:

- UNMOUNT

Restrictions

1. You must have *IOSYSCFG special authority to use this command.

For more information about the RMVMFS and UNMOUNT commands and the associated parameters and options, see the CL topic in the iSeries Information Center.

RMVMFS/UNMOUNT Display

```
Remove Mounted FS (RMVMFS)

 Type choices, press Enter.

 Type of file system  . . . . . > *NFS   *NFS, *UDFS, *NETWARE, *ALL
 Directory mounted over . . . . . > '/USER/WORK'

 Mounted file system  . . . . . > '/CLASSES/CLASS1'









                                                                      Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 42. Using the Remove Mounted FS (RMVMFS) display

When you use the RMVMFS or UNMOUNT commands, you can specify many parameters and options:

- The TYPE parameter on the ADDMFS command specifies the type of file system being unmounted.
- The MNTOVRDIR parameter on the ADDMFS command specifies the path name of the directory that was mounted over ("covered") by a previous ADDMFS command.
- The MFS parameter on the ADDMFS command specifies the file system to be unmounted.

  Note: This parameter can only be used to unmount a block special file (*BLKSF object) when you specify the *UDFS value in the TYPE parameter on the ADDMFS command.

Examples

Example 1: Unmounting a Directory.

```
RMVMFS  TYPE (*NFS) MNTOVRDIR('/tools')
```

This command unmounts a Network File System that is accessible on directory */tools*.

Example 2: Unmounting a User-Defined File System.

```
RMVMFS  TYPE(*UDFS) MFS('/DEV/QASP02/A.udfs')
```

This command unmounts the user-defined file system `/DEV/QASP02/A.udfs`.

Example 3: Unmounting all mounted file systems on a client.

```
RMVMFS  TYPE(*ALL) MNTOVRDIR(*ALL)
```

This command unmounts all the file systems that a client has mounted.

Example 4: Unmounting all mounted file systems on a specific client directory.

```
RMVMFS  TYPE(*ALL) MNTOVRDIR('/napa')
```

This command unmounts all the file systems that a client has mounted over `/napa`.

DSPMFSINF (Display Mounted File System Information) Command

Purpose

The Display Mounted File System Information (DSPMFSINF) command displays information about a mounted file system.

Users can also issue this command by using the following alternative command name:

- STATFS

For more information about the DSPMFSINF command and the associated parameters and options, see the CL topic in the iSeries Information Center.

DSPMFSINF/STATFS Display

```
Display Mounted FS Information (DSPMFSINF)

 Type choices, press Enter.

 Object . . . . . .   /dev/qasp02/kate.udfs

 Output  . . . . . .  *           *, *PRINT









                                                                      Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 43. Using the Display Mounted FS Information (DSPMFSINF) display

When you use the DSPMFSINF command, you only have to specify one parameter:

- The required OBJ parameter **1** on the DSPMFSINF command specifies the path name of an object that is within the mounted file system whose statistics are to be displayed. Any object in the mounted file system can be specified. For example, it can be a directory (*DIR) or a stream file (*STMF).

When you use the DSPMFSINF or STATFS command, you will view a series of displays like the two that follow:

```
Display Mounted FS Information

 Object . . . . . :   /home/students/ann

 File system type . . . . . :   User-defined file system

 Block size . . . . . :   4096
 Total blocks . . . . . :   2881536
 Blocks free  . . . . . :   1016026
 Object link maximum  . . . . . :   32767
 Directory link maximum . . . . :   32767
 Pathname component maximum . . :   510
 Path name maximum  . . . . . :   No maximum
 Change owner restricted  . . . :   Yes
 No truncation  . . . . . :   Yes
 Case Sensitivity . . . . . :   No



                                                                      More...
 Press Enter to continue.

 F3=Exit   F12=Cancel
 (C) COPYRIGHT IBM CORP. 1980, 1996.
```

Figure 44. Display Mounted FS Information (DSPMFSINF) output (1/2)

This first display shows basic information about a mounted file system.

```
Display Mounted FS Information

 Path of mounted file system  . :   /dev/qasp02/kate.udfs

 Path mounted over  . . . . . :   /home/students/ann

 Protection . . . . . :   Read-write
 Setuid execution . . . . . :   Not supported
 Mount type . . . . . :   Not supported
 Read buffer size . . . . . :   Not supported
 Write buffer size  . . . . . :   Not supported
 Timeout  . . . . . :   Not supported
 Retry Attempts . . . . . :   Not supported
 Retransmission Attempts  . . . :   Not supported
 Regular file attribute minimum
   time . . . . . :   Not supported
 Regular file attribute maximum
   time . . . . . :   Not supported

                                                                      More...
 Press Enter to continue.

 F3=Exit   F12=Cancel
```

Figure 45. Display Mounted FS Information (DSPMFSINF) output (2/2)

You can see from this display advanced types of information are not supported for user-defined file systems.

Examples

Example 1: Displaying Statistics of a Mounted File System.

```
DSPMFSINF OBJ('/home/students/ann')
```

This command displays the statistics for the mounted file system that contains /home/students/ann.

Example 2: Displaying '/QSYS.LIB' File System Statistics.

```
DSPMFSINF  OBJ('/QSYS.LIB/MYLIB.LIB/MYFILE.FILE')
```

This command displays the statistics for the /QSYS.LIB file system that contains *FILE object MYFILE in library MYLIB.

Chapter 6. Using the Network File System with iSeries File Systems

There are several exceptions to using iSeries file systems with NFS on various clients. This is because you are able to export several different file systems on an iSeries NFS server. Each file system has its own set of requirements and deviations through NFS from its normal functioning state. The purpose of this chapter is to make you aware of these differences for the specific file system you are accessing through NFS.

With OS/400 Version 4 Release 4 (V4R4), the following file systems received enhancements to support stream files larger than 2 gigabytes:

- Library File System (QSYS.LIB)
- Open Systems File System (QOpenSys)
- "Root" (/)
- User-Defined File System (UDFS)

NFS on iSeries also supports these large files in V4R4 with the exception of byte-range locking. The fcntl() API will only function with these files from an NFS client if the byte-range locking sizes and offsets fall under the 2 GB limit.

For detailed information on the file systems that NFS supports and large file support, refer to Large file support for integrated file system APIs in the **File systems and management** topic of the iSeries Information Center.

″Root″ File System (/)

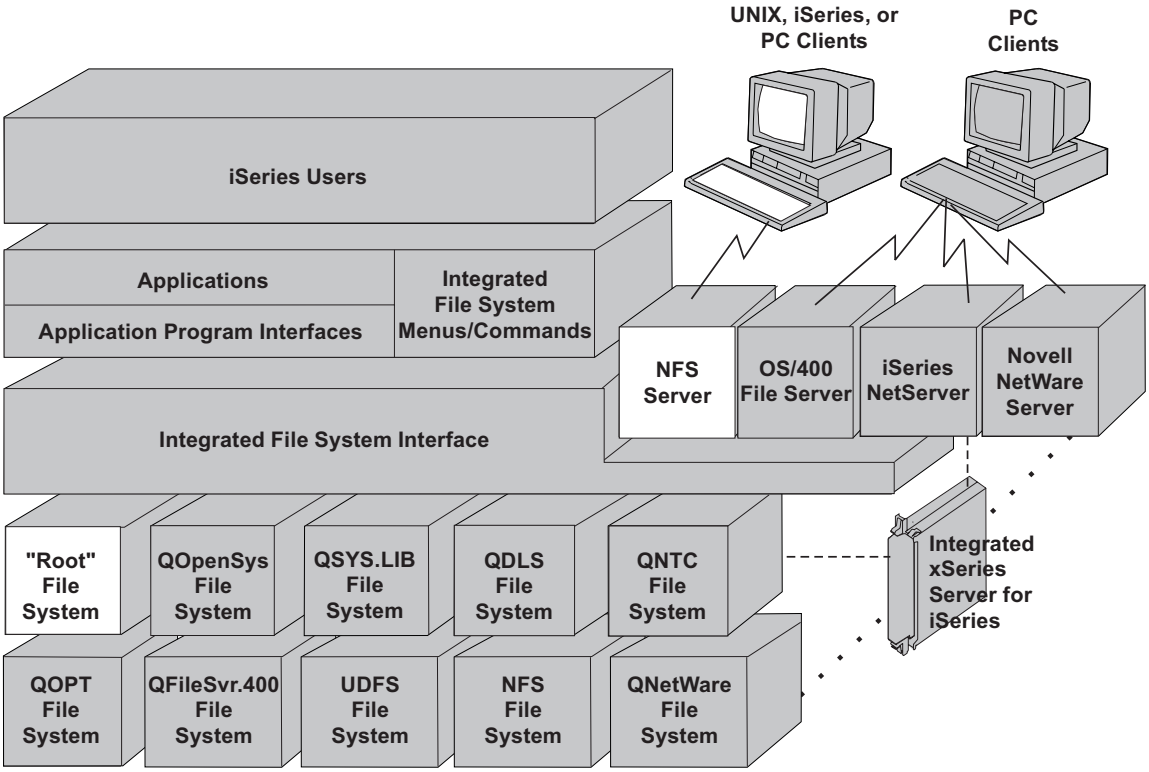

Figure 46. The ″Root″ (/) file system accessed through the NFS Server

Network File System Differences

Case-Sensitivity

When a remote UNIX client mounts an object that the server exports from the “root” (/) file system, it will always function as case-insensitive.

Read/Write Options

No matter what options the client specifies on the MOUNT command, some server file systems from “root” (/) exist as only read-write. However the client mounts a file system determines how the file system is treated and how it functions on the client.

Open Systems File System (QOpenSys)

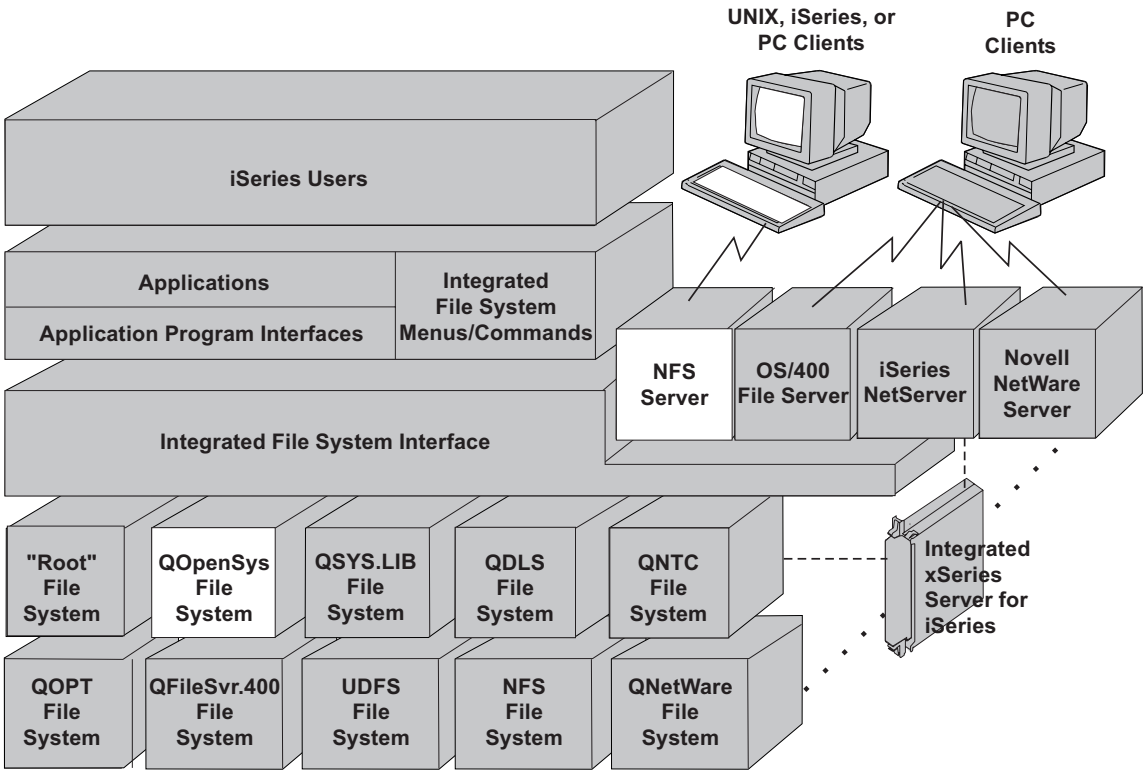

Figure 47. The QOpenSys file system accessed through the NFS Server

Network File System Differences

Case-Sensitivity

When a remote UNIX client mounts an object that the server exports from the QOpenSys file system, the object will always function as case-sensitive.

Read/Write Options

No matter what options the client specifies on the MOUNT command, some server file systems from QOpenSys exist as read-only or read-write. However the client mounts a file system determines how the file system is treated and how it functions on the client.

Library File System (QSYS.LIB)

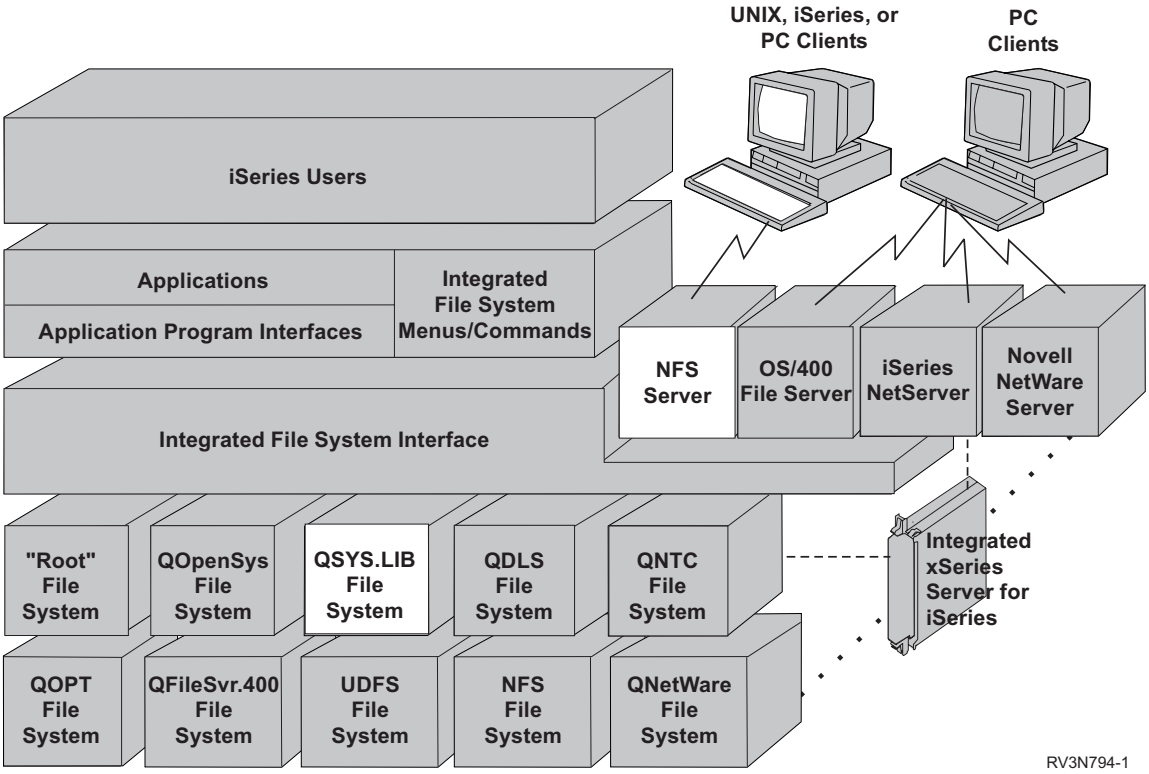

Figure 48. The QSYS.LIB file system accessed through the NFS Server

Network File System Differences

Exporting and QSYS.LIB

You can export some .LIB and .FILE objects. If you export .SAVE files and a client mounts them, then all attempts to open the files will fail. In general, you should export only things that clients need access to.

All object types in the QSYS.LIB file system can be exported and mounted successfully without error. All operations on exported and mounted file systems will function as if the object existed locally on the server. For example, the only object types in the QSYS.LIB file system that support all file input/output (I/O) operations are database members (.MBR) and user spaces (.USRSPC).

NFS always opens data files in binary mode. If you are reading source physical file members, you may not see line feeds or carriage returns. Use the CPYTOIMPF command first in order to view the line feeds and carriage returns of the file member.

QPWFSERVER Authorization List

The QPWFSERVER is an authorization list (object type *AUTL) that provides additional access requirements for all objects in the QSYS.LIB file system being accessed through remote clients. The authorities specified in this authorization list apply to all objects within the QSYS.LIB file system.

The default authority to this object is PUBLIC *USE authority. The administrator can use the EDTAUTL (Edit Authorization List) or WRKAUTL (Work With

Authorization List) commands to change the value of this authority. The administrator can assign PUBLIC *EXCLUDE authority to the authorization list so that the general public cannot access QSYS.LIB objects from remote clients.

Mounting and QSYS.LIB

Users can mount the QSYS.LIB file system on a client, but users cannot mount over the QSYS.LIB file system. Users should export and mount a sub-library from QSYS.LIB rather than mounting QSYS.LIB directly on the client. The reason for this is that QSYS.LIB contains hundreds of objects. Trying to display or process all of the objects can affect client performance.

Support for User Spaces

NFS supports the exporting and mounting of user spaces, with the following exceptions:

- User spaces cannot be over 16 megabytes
- User spaces are not CCSID-tagged or tagged for a code page by default. If a CCSID is asked for, NFS will translate the data if the user specifies the CCSID.
- User space files (*USRSPC object type) can be appended to using NFS, but this may produce unpredictable results with how data is written.

File Modes of Database Members

The file mode of any database members needs to be the same as the file mode of the parent file. No matter what users specify, NFS will always create new database members with the file mode of the parent. If users specify other file modes than that of the parent, they will not receive an error return code. NFS will create the new member with the file mode of the parent no matter what the user specifies.

Path Names of .FILE Objects

Users need to be aware of the maximum database record length when editing database members. Users specify the record length when creating a physical file. The default record length for all .FILE objects created in QSYS.LIB is 92 bytes if created with one of the following methods:

1. mkdir() API
2. MKDIR (Make Directory) command
3. MD (Make Directory) command
4. CRTDIR (Create Directory) command

Source physical files contain a date stamp (6 bytes) and sequence number (6 bytes), using 12 total bytes of data. This is accounted for by subtracting 12 bytes from 92, which leaves a default of 80 bytes per record for source physical files. For any record length specified, the real amount of bytes per record is the number specified minus 12 bytes for source physical files.

Byte-Range Locks

QSYS.LIB does not support byte-range locking. The fcntl() API will fail with error condition ENOSYS if used by clients.

Case-Sensitivity

QSYS.LIB is case-insensitive. UNIX clients are typically case-sensitive. How users read directories and match patterns will determine which files the system displays when displaying QSYS.LIB through a UNIX client.

For example, there is one file in a given directory, `AMY.FILE`. The UNIX LS (list) command will display all the contents of the directory. When users issue the following command:

```
ls a*
```

The system will display no files or objects.

However, when users issue this command:
`ls A*`

The system will display `AMY.FILE`

Pattern matching occurs in the client, not the server, and all entries come from QSYS because NFS only reads directories, but does not match patterns. The command `ls annie.file` will work because it does not rely on pattern-matching. Quoted (extended) names are returned exactly as they are stored, as they are case-sensitive.

Document Library Services File System (QDLS)

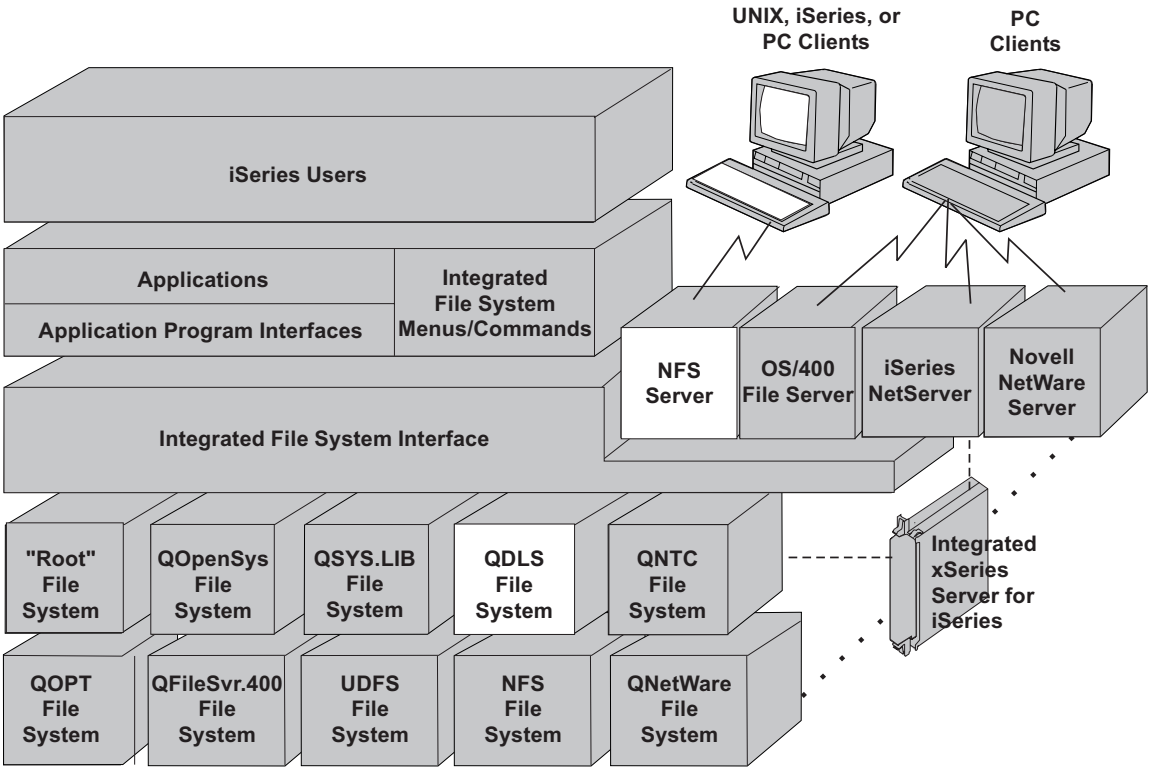

Figure 49. The QDLS file system accessed through the NFS Server

Network File System Differences

Mounting and QDLS
Users can mount the QDLS file system on a client, but users cannot mount over the QDLS file system.

File Creation
Users cannot create regular files in the top-level `/QDLS` directory. Users can only create files in the sub-directories of `/QDLS`.

Path Name Length
The name of any QDLS component can be up to 8 characters long, and the extension (if any) can be up to 3 characters long. The maximum length of the path

name is 82 characters, assuming an absolute path name beginning with /QDLS. When mounting the QDLS file system on a client, the complete path name cannot exceed 87 characters (including /QDLS).

Anonymous Users

Clients can mount the QDLS file system through the NFS server. If anonymous clients plan to use objects within QDLS, however, they must first register with the document library services through enrollment. Administrators can enroll QNFSANON or other users with the QDLS Folder Management Services (FMS) by using the ADDDIRE (Add Directory Entry) command. All anonymous client requests that are mapped to QNFSANON will fail at the server if you do not enroll the QNFSANON user profile in FMS.

For more information regarding the QDLS file system, see

- Document library services file system (QDLS) topic in the iSeries Information Center

Optical File System (QOPT)

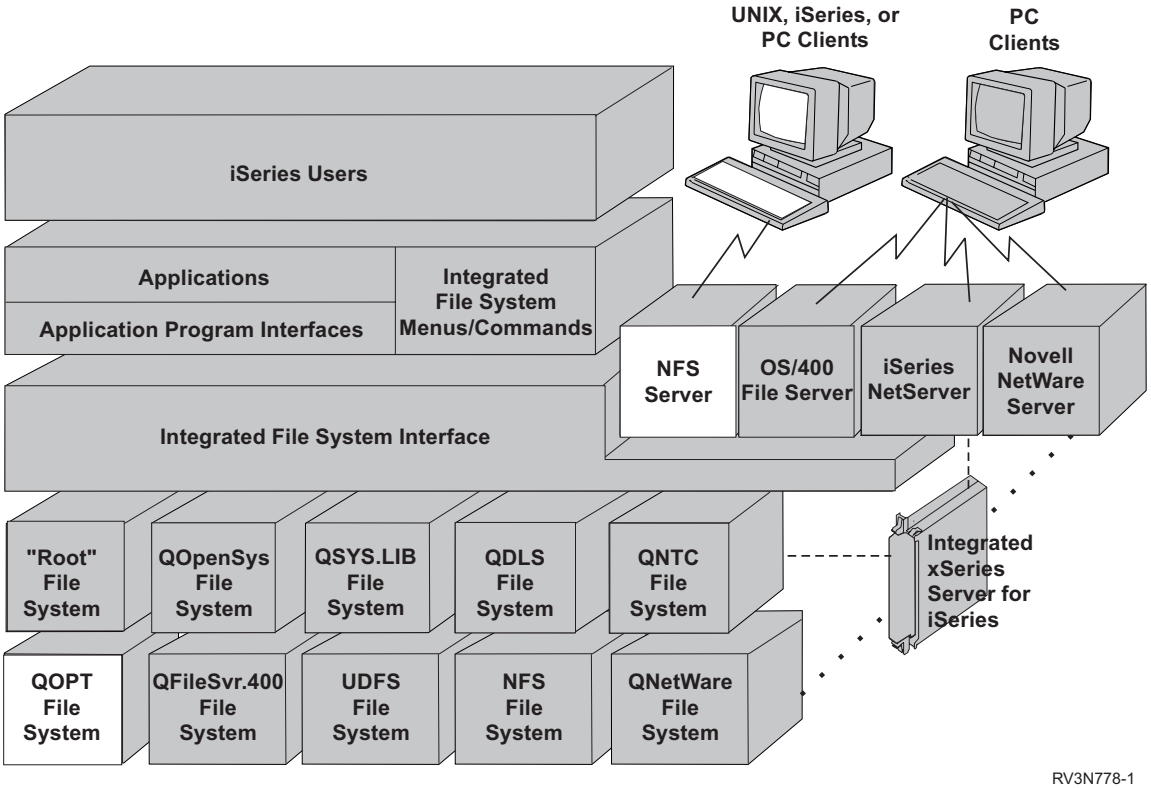

Figure 50. The QOPT file system accessed through the NFS Server

Network File System Differences

Mounting and QOPT

Users can export QOPT file system and mount it on a client. You cannot mount over the QOPT file system. Due to the statelessness of NFS and the fact that optical storage can not be reused unless the entire optical volume is re-initialized, the QOPT file system will be treated as a read-only file system once exported and mounted on a client. Native users on the server will continue to be able to write to the QOPT file system.

Note: When exporting any path in the QOPT file system, you must specify the read-only (RO) option. Otherwise, the export request will fail.

Case-Sensitivity

QOPT is case-insensitive. It converts lowercase English alphabetic characters to uppercase when used in object names. Therefore, the path name `/QOPT/volume/dir/file` represents the same path as `/QOPT/VOLUME/DIR/FILE`..

Security and Authorization

The QOPT file system offers **volume-level security**, as opposed to file or directory-level security. Each optical volume is secured by an authorization list. If a user needs access to directories or files on a volume, they will need access to the optical volume. The system administrator, or a user with authorization list management authority, can grant access by doing the following:

1. Use the Work with Optical Volumes (WRKOPTVOL) command to find out which authorization list secures the volume.
2. Use the Edit Authorization List (EDTAUTL) command to add the user to the authorization list.

For more information on optical security, see the Optical Support book.

User-Defined File System (UDFS)

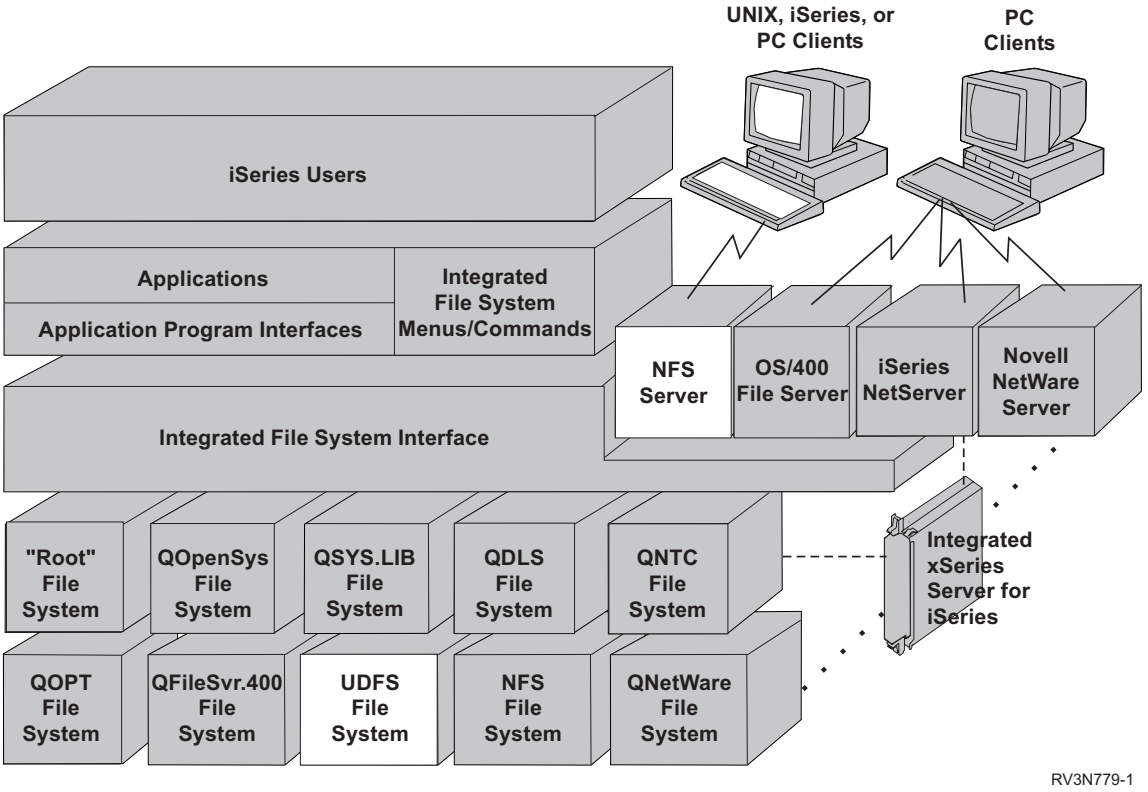

Figure 51. The UDFS file system accessed through the NFS Server

Network File System Differences

Case-Sensitivity

When remote UNIX clients mount objects that the server exports from a UDFS, the case-sensitivity is variable, depending on how the user created the UDFS. A UDFS that is mounted on a UNIX client can cause the case-sensitivity to change in the middle of a directory tree.

System and User Auxiliary Storage Pools

The contents of a UDFS exist in a user auxiliary storage pool (ASP), but the UDFS block special file itself always lies in the system ASP.

Administrators of UNIX Clients

Network File System Differences

Directory Authority

UNIX clients need *X (execute) authority to create files within a directory. This is true when working with any mounted iSeries file system. By default, UNIX clients may not create directories with *X (execute) authority attached. UNIX clients may need to change the UMASK mode bits to ALL on the execute bit when directories are created. This will allow objects to be created within that directory.

Chapter 7. NFS Startup, Shutdown, and Recovery

NFS startup performs separately and independently on each machine. The startup of an NFS component on one server does not trigger the startup of an NFS component on another server. For example, if you start the Network Lock Manager on a client, the NLM on the server will not automatically start up. For proper functioning of NFS, there is an implied order in which the daemons should be started on any given server or network.

For example, the NFS server daemon must be operational on a given machine before an NFS client on another server can successfully issue any requests to that server. Similarly, the mount daemon on a given server must be operational before another machine can successfully issue a mount request to that daemon.

Administrators have the ability to start and end NFS servers and clients at any time, in any order. This action will not cause error messages to be issued. However, it is recommended that users should start and end NFS in the order that is specified below for best operation.

Note: Do not start File Server Support/400 (FSS/400) and NFS at the same time. Only one of these server applications can operate on iSeries at any given time. If FSS/400 is operating at the time of NFS startup, the RPC Binder Daemon (port mapper) will fail to connect to port 111. Furthermore, the RPC daemon will not be able to assign any ports to server processes. The use of NFS is recommended.

Configuring TCP/IP

You must install and properly configure TCP/IP prior to starting NFS support for the first time. NFS uses the system configuration to determine the local server name. There must be an entry in the hosts table for both the ″short″ and ″long″ name, unless you are using a Domain Name Server (DNS). Perform the following steps to correctly configure TCP/IP and its options:

1. Go to the **Configure TCP/IP** menu (CFGTCP).
2. Select option **12** (Change TCP/IP Domain or CHGTCPDMN).

   You can find the short name in the Host name parameter. The long name is the short name appended by a '.' and the Domain name (the next parameter). For example, `ASHOST01` is a short name, and `ASHOST01.NETWORK.DOMAIN.COM` is a long name.
3. Update information if needed.

   The Host name search priority tells the system the order in which to try to resolve host names. *LOCAL means to search the TCP/IP host table first, and *REMOTE says to use a DNS first at the specified IP address. If you do not need to update information on this screen, press PF12.
4. Select option **10** (Work with TCP/IP Host Table Entries).
5. Verify that there is both a long and short host name entry for the IP address of the local system. If you do not know this address, select option 1 from the CFGTCP menu. Additionally, you can select one of the two following options:
   a. Select option 2 (Change) from the CFGTCP menu to add a name for an address.

b. Select option 1 from the CFGTCP menu to add an entire new address with names.

6. Verify that the names LOOPBACK and LOCALHOST are associated with the IP address 127.0.0.1 in the host table.
7. Verify that the long and short names of each NFS server you need access to are included in the host table. You only need to do this if you will be using the system as an NFS client. If they are not in the host table, and a DNS is not being used, then you must add the long and short names.

Here is an example of a TCP/IP Host table:

```
     Internet          Host
Opt  Address           Name

     19.45.216.4       THISHOST
                       THISHOST.COMPANY.DOM1.COM
     19.45.216.93      NFSSERV1
                       NFSSERV1.COMPANY.DOM1.COM
     127.0.0.1         LOOPBACK
                       LOCALHOST
```

Implications of Improper Startup and Shutdown

The various components of the Network File System make up a complex and interdependent system. This means that they rely on each other for common, efficient functioning. There are, therefore, many functions that can fail if users startup or shutdown improperly:

- If the user does not start the RPC binder daemon first, then all local server ports will fail to map properly. No requests from the client will be processed.
- If the client starts before the server has loaded the mount daemon, any mount requests the client makes will fail. Furthermore, the NLM and NSM daemons will not be able to grant locks until users start them on both the client *and* the server.
- Should the client be started and make mount requests before the server has issued the export command, all mount requests will fail.
- If a user ends the RPC binder daemon (port mapper) first, then all the other daemons cannot unregister with the RPC binder (port mapper) daemon.
- If users do not end all the daemons, then there can be confusion on the next startup of which daemons are operating and which are not.

Proper Startup Scenario

In an NFS **server** startup:

1. The user starts the RPC binder (port mapper) daemon (QNFSRPCD). This daemon then waits on a known port (#111) for local RPC requests to register a service. This daemon also waits for remote RPC requests to query a local service.
2. The user calls the export command, creating a list of exported directories in the export table from information contained in the /etc/exports file.
3. The user starts the NFS server daemon (QNFSNFSD) or daemons. It registers to the local RPC binder daemon, which knows on which port the NFS server waits for requests (the standard is #2049). All server daemons will use this same port. The NFS server daemons then wait on the port for RPC requests from NFS clients to access local files.

4. The user starts the mount daemon (QNFSMNTD). This daemon registers to the local RPC binder daemon. It then waits on the assigned port for RPC requests from NFS clients to mount local file systems.
5. The user starts the NSM daemon (QNFSNSMD). It registers to the local RPC binder daemon. It then waits on the assigned port for RPC requests to monitor systems.
6. The user starts the NLM daemon (QNFSNLMD). It registers to the local RPC binder daemon. It then waits on the assigned port for RPC requests to manage locks.

In a typical startup, if you specify *ALL for the SERVER parameter for the Start Network File System Server (STRNFSSVR) command, you will automatically start all the daemons in the correct order.

In an NFS **client** startup:

1. The user starts the RPC binder (port mapper) daemon, if it is not already operational. On a given system, a single port mapper is used for both client and server.
2. The user starts the block I/O daemon (QNFSBIOD) or daemons. This daemon controls the caching of data and attributes that have been transmitted from the server.
3. The user starts the NSM daemon, if it is not already operational. On a given system, a single NSM operates for both the client and server.
4. The user starts the NLM daemon, if it is not already operational. On a given system, a single NLM is used for both the client and server.

STRNFSSVR (Start Network File System Server) Command

Purpose

The Start Network File System Server (STRNFSSVR) command starts one or all of the Network File System (NFS) server daemons.

You should use the SERVER(*ALL) option, which will start the daemons in the following order, as well as call the export command. This order is the recommended order for starting the Network File System.

- The Remote Procedure Call (RPC) binder daemon
- The block I/O (BIO) daemon
- Call the export command
- The server (SVR) daemon
- The mount (MNT) daemon
- The network status monitor (NSM) daemon
- The network lock manager (NLM) daemon

If you are choosing to start just one daemon, be sure you understand the appropriate order for starting NFS daemons and the possible consequences of starting daemons in an order other than that specified above.

If you attempt to start a daemon or daemons that are already running, they will not cause the command to fail, and it will continue to start other daemons you have requested to start. The command will issue diagnostic message CPDA1BA if the daemon is already running. For best results, end NFS daemons before attempting the STRNFSSVR command.

Displaying NFS Server Daemons

To display NFS server daemons, you can use the Work with Active Jobs (WRKACTJOB) command and look in the subsystem QSYSWRK for the existence of the following jobs:

- QNFSRPCD, the RPC Binder Daemon (RPCD)
- QNFSNFSD, the NFS Server Daemon (NFSD, there may be multiple entries for this daemon)
- QNFSMNTD, the Mount Daemon (MNTD)
- QNFSNSMD, the Network Status Monitor Daemon (NSMD)
- QNFSNLMD, the Network Lock Manager Daemon (NLMD)

Status Consideration

When displaying the status of NFS server daemons (NFSD) using the WRKACTJOB (Work with Active Jobs) command, there may be different status values listed. The status of the first NFSDs not in use will be TIMW and all other NFSDs will be listed as MTXW.

Restrictions

1. You must have *IOSYSCFG special authority to use this command.
2. You must be enrolled in the system distribution directory. Use the ADDDIRE command to enroll in the system distribution directory.
3. To use the STRNFSSVR command, you must first have TCP/IP operating on iSeries.

For more information about the STRNFSSVR command and its parameters and options, see the CL topic in the iSeries Information Center.

STRNFSSVR Display

```
Start NFS Server (STRNFSSVR)

 Type choices, press Enter.

 Server daemon  . . . . . . . . . . > *ALL           *ALL, *RPC, *BIO, *SVR...
 Number of server daemons . . . .   1               1-20 server daemons
 Number of block I/O daemons  . .   1               1-20 server daemons
 Timeout for start of daemon  . .   *NOMAX         1-3600 seconds

                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 52. Using the Start NFS Server (STRNFSSVR) display

When you use the STRNFSSVR command, you can specify many parameters:

- The required SERVER parameter on the STRNFSSVR command specifies the Network File System daemon jobs to be started by this command. The specified daemon should not already be running.

- The NBSVR parameter on the STRNFSSVR command specifies the number of NFS server (*SVR) daemon jobs you want to have running. Additional daemons will be started if the number you specify on this parameter is greater than the number of server daemons already running on the server. This parameter can only be used if you specify SERVER(*SVR).
- The NBRBIO parameter on the STRNFSSVR command specifies the number of NFS block I/O (*BIO) daemon jobs you want to have running. Additional daemons will be started if the number you specify on this parameter is greater than the number of block I/O daemons already running on the server. This parameter can only be used if you specify SERVER(*BIO).
- The STRJOBTIMO parameter on the STRNFSSVR command specifies the number of seconds to wait for each daemon to successfully start. If a daemon has not started within the timeout value, the command will fail.

Examples

Example 1: Start All NFS Daemons

```
STRNFSSVR  SERVER(*ALL) STRJOBTIMO(*NOMAX)
```

This command starts all NFS daemons, and waits forever for them to start. No daemons should be previously running.

Example 2: Start Only One Daemon

```
STRNFSSVR  SERVER(*MNT)
```

This command starts the NFS mount daemon, and waits up to the default of 30 seconds for it to start. The mount daemon should not be already running, and other daemons have been started in the appropriate order.

Proper Shutdown Scenario

Shutting down an NFS server properly allows for all jobs to finish and all requests to be completed. In general, the order of actions required for the server to shutdown are the exact opposite of actions required for the **server** to startup:

1. The user ends the NLM daemon (QNFSNLMD).
2. The user ends the NSM daemon (QNFSNSMD). All locks that are held on local files by remote client applications are disengaged.
3. The user ends the mount daemon (QNFSMNTD). All remote client mounts of local file systems are disengaged.
4. The user ends the NFS server daemon (QNFSNFSD) or daemons.
5. The user ends the RPC binder (port mapper) daemon (QNFSRPCD).

If you specify `*ALL` for the SERVER parameter for the End Network File System Server (ENDNFSSVR) command will automatically end all the daemons in the correct order.

The order of **client** shutdown processes is generally the opposite from which the user starts the processes.

1. The user ends the NLM daemon, if it exists on the client. A single server NLM can operate for both the client and server.
2. The user ends the NSM daemon, if it exists on the client. A single server NSM can operate for both the client and server.
3. The user ends the block I/O daemon (QNFSBIOD) or daemons.
4. The RPC binder (port mapper) daemon is ended.

Shutdown Consideration

TCP/UDP Timeout Conflict

When ending the NFS server, the socket port closes. If the NFS server is immediately re-started, then the server may not be able to connect to the socket port. The underlying TCP/IP support on iSeries renders this port unavailable for a short period. If you wait for a short period before re-starting the NFS server, then it will connect to the socket port as usual.

ENDNFSSVR (End Network File System Server) Command

Purpose

The End Network File System Server (ENDNFSSVR) command ends one or all of the Network File System (NFS) server daemons.

You should use SERVER(*ALL), which will end the daemons in the following order. (This order is the recommended order for ending the Network File System daemons.)

- The network lock manager (NLM) daemon
- The network status monitor (NSM) daemon
- The mount (MNT) daemon
- The server (SVR) daemon
- The block I/O (BIO) daemon
- The Remote Procedure Call (RPC) binder daemon

If you are choosing to end just one daemon, be sure you understand the appropriate order for ending NFS daemons and the possible consequences of ending daemons in an order other than that specified above.

If you attempt to end a daemon or daemons that are not running, they will not cause the command to fail, and it will continue to end other daemons you have requested to end.

Displaying NFS Client Daemons

To display NFS client daemons, you can use the Work with Active Jobs (WRKACTJOB) command and look in the subsystem QSYSWRK for the existence of the following jobs:

- `QNFSRPCD`, the RPC Binder Daemon (RPCD)
- `QNFSMNTD`, the Mount Daemon (MNTD)
- `QNFSNSMD`, the Network Status Monitor Daemon (NSMD)
- `QNFSNLMD`, the Network Lock Manager Daemon (NLMD)
- `QNFSBIOD`, the Block I/O Daemon (BIOD, there may be multiple entries for this daemon)

Restrictions

1. You must have *IOSYSCFG special authority to use this command.

For more information about the ENDNFSSVR command and its parameters and options, see the CL topic in the iSeries Information Center.

ENDNFSSVR Display

```
End NFS Server (ENDNFSSVR)

 Type choices, press Enter.

 Server daemon  . . . . . .  *ALL     *ALL, *RPC, *BIO, *SVR...
 Timeout for end of daemon  . . .  30      1-3600 seconds

                                                                    Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 53. Using the End NFS Server (ENDNFSSVR) display

When you use the ENDNFSSVR command, you can specify many parameters:

- The required SERVER parameter on the ENDNFWSVR command specifies the Network File System daemon jobs to end.
- The ENDJOBTIMO parameter on the ENDNFSSVR command specifies the number of seconds to wait for each daemon to successfully end. If a daemon has not ended within the timeout value, the command will fail.

Examples

Example 1: End All Daemons

```
ENDNFSSVR  SERVER(*ALL)
```

This command ends all NFS daemon jobs that are running.

Example 2: End a Single Daemon

```
ENDNFSSVR  SERVER(*MNT) ENDJOBTIMO(*NOMAX)
```

This command ends the NFS mount daemon, and waits forever for it to end. The mount daemon was previously running, and other daemons have been ended in the appropriate order.

Start or stop NFS in iSeries Navigator

NFS servers can also be managed using iSeries Navigator. To start or stop NFS server daemons, perform the following steps:

1. In **iSeries Navigator**, expand your server.
2. Expand **Network**.
3. Expand **Servers**.
4. Click **TCP/IP**. The right panel displays a list of TCP/IP servers.
5. Right-click **NFS** and select **Start** or **Stop**, depending on which task you want to perform. You can start or stop all the NFS server daemons, or start or stop them individually.

   The following figure displays these steps. In the figure, NFS has a status of stopped.

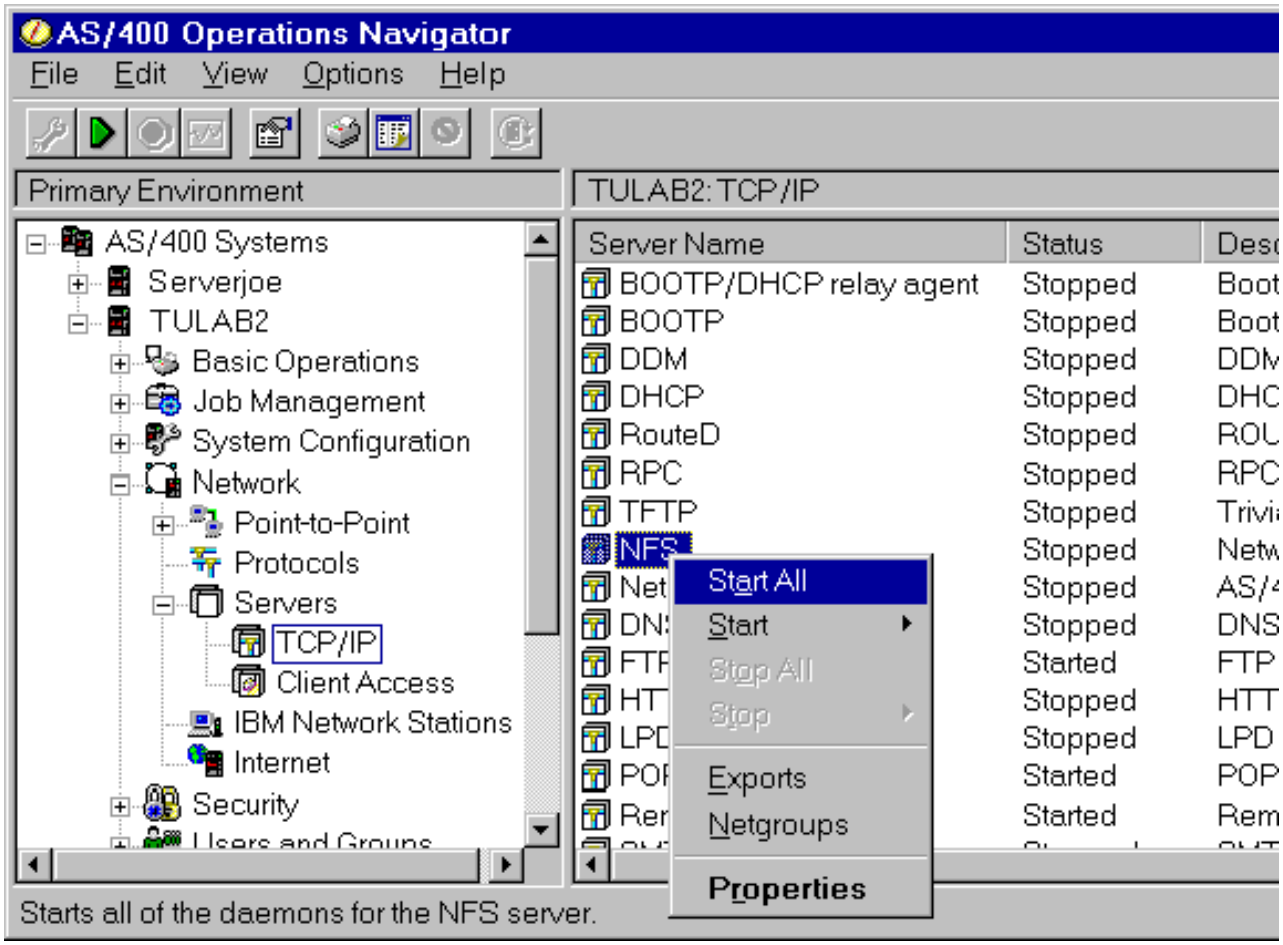

Figure 54. Starting or stopping NFS server daemons.

You can also display the status of each individual daemon by choosing **Properties** when you right-click on **NFS**. This brings up the following dialog box:

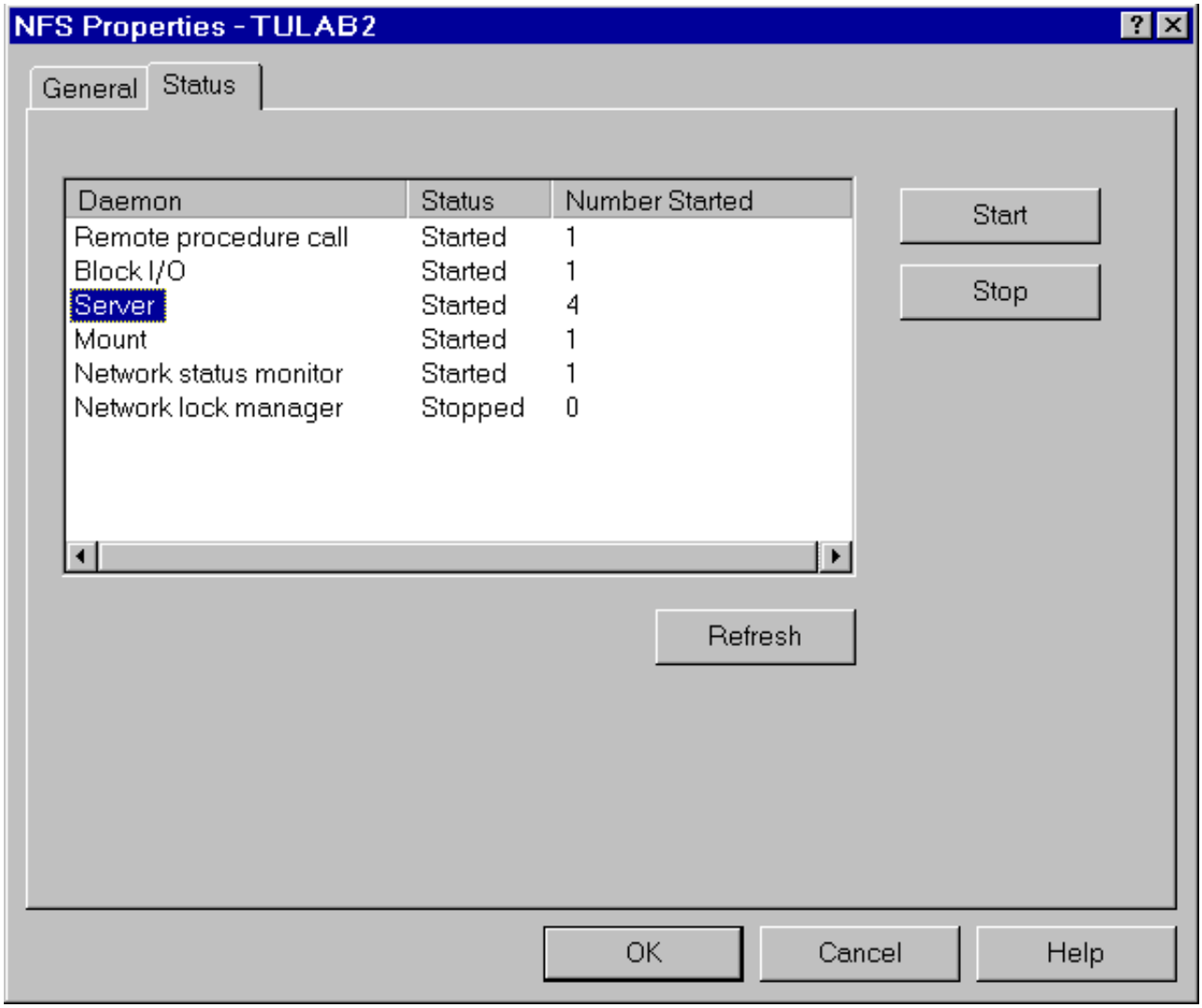

Figure 55. NFS Properties dialog box.

In the example, Chris Admin has decided to start 4 of the *Server* type daemons to give better throughput. You can start up to 20 of these daemons from the *General* tab of the previous dialog box. Notice that the Network lock manager daemon is stopped. This could indicate that it encountered a problem by trying to start up. Alternately, it could mean that the administrator chose to end it specifically because of no need for byte range locking.

Both NFS and RPC share the same Remote Procedure Call Binder daemon. Starting or stopping NFS will start or stop RPC, and vice versa. For example, stopping the RPC server will end the Remote procedure call NFS daemon and may cause NFS to stop functioning correctly.

Locks and Recovery

Clients can also introduce locks onto mounted server file systems. Locks will give a user shared or exclusive access to a file or part of a file. When a user locks a file, any process requiring access to the file must wait for the lock to be released before processing can continue.

There are two kinds of locks that clients can establish on all or part of a file: **exclusive** and **shared**. When a client obtains an exclusive lock, no other processes can obtain a lock on that portion of the file. When a client obtains a shared lock, other processes can obtain a shared lock to the same portion of the file.

Why Should I Lock a File?

A user or application can use byte-range locks through NFS to improve NFS performance. Because the NFS protocol is stateless, a given client may not be aware of changes made on the server (due to client caches). Locking ensures that this problem will not occur during critical times when the server updates files.

How Do I Lock A File?

An application at the client, controlled by the user, can start a lock request against a remote file that is mounted with NFS. The client will then send the operation to its local network lock manager Daemon through an RPC request. The client-side NLM daemon will then forward the request to the corresponding server-side NLM through RPC.

Users can call the fcntl() API to lock parts of files and specify lock parameters. For a complete description of the fcntl() API, see the API topic in the iSeries Information Center.

The server NLM daemon will then perform the lock operation against the corresponding file. The server NLM daemon generates a response that the client NLM daemon receives. The client NLM daemon generates a response for the NFS client with the results of the lock request. The NFS client will then return the result to the user through the application.

Stateless System Versus Stateful Operation

The statelessness of the Network File System does not integrate well with the statefulness of file locks. Problems can occur in releasing locks if either the client or server fails while their respective daemons hold a lock or locks.

If a client with a granted lock request should happen to fail, a specific set of operations will occur at startup time to recover the locks:

1. When the user restarts the NSM daemon on a system, the daemon will send a change of state RPC to other NSM daemons in the network. This message is transmitted only to the other NSM daemons that the failed system is aware of in the network.
2. After receiving a change of state from a remote system, the NSM uses RPC to communicate with the local NLM. The NSM informs the NLM that a lock was held at the time of system failure.
3. If the failed system is a server, then after recovery, the local server NLM attempts to re-establish file locks that client processes might have held. The information about which remote clients held locks on the server is stored in the local file `/etc/statd`. Users should not edit or change this file in any way. Improperly changing this file may affect recovery from a server failure.

When the local system is a server, and the client never restarts, then an infinite backlog of requests will build up for the locked file. This file will *never* be released from its lock. This can be averted with the Release Integrated File System Locks (RLSIFSLCK) command.

RLSIFSLCK (Release Integrated File System Locks) Command

Purpose

The Release Integrated File System Locks (RLSIFSLCK) command can be used to release all Network File System (NFS) byte-range locks. These locks might be held by a specified NFS client, or on a particular specified object. This command should only be used to free resources that cannot be freed using normal means. When locks are released with this command, a successful return code is sent to the locking application.

For more information about byte-range locks and a complete description of the fcntl() API, see the API topic in the iSeries Information Center.

Restrictions

1. You must have *IOSYSCFG special authority to use this command.

For more information about the RLSIFSLCK command and associated parameters and options, see the CL topic in the iSeries Information Center.

RLSIFSLCK Display

```
Release File System Locks (RLSIFSLCK)

 Type choices, press Enter.

 Remote location  . . . . . .   TULAB2

 Object . . . . . .

                                                                      Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 56. Using the Release File System Locks (RLSIFSLCK) display

When you use the RLSIFSLCK command, you can specify many parameters:

- The RMTLOCNAME parameter on the RLSIFSLCK command specifies the host name or internet address of a remote system whose NFS-related locks on local files are to be released.
- The OBJ parameter on the RLSIFSLCK command specifies the path name of an object on which all byte-range locks are to be released. This will release all locks on that object, regardless of the type of lock or the type of process that holds them.

Examples

Example 1: Releasing Locks for a Remote Client.

```
RLSIFSLCK RMTLOCNAME('TULAB2')
```

This command releases the NFS-related locks that are held on local files by the system TULAB2.

Example 2: Releasing Locks for a Local Object.

```
RLSIFSLCK OBJ('/classes/class2/proj3')
```

This command releases all byte-range locks held on the object `/classes/class2/proj3`.

Chapter 8. Integrated File System APIs and the Network File System

Error Conditions

There are two error conditions that commonly appear when working with the Network File System through integrated file system APIs (application program interface) that require special consideration. These error conditions are the ESTALE and EACCES error conditions.

ESTALE Error Condition

The return of this error number means that the server has rejected the file or object handle.

If you are accessing a remote file through the Network File System, the file may have been deleted at the server. Or, the file system that contains the object may no longer be accessible.

EACCES Error Condition

The return of this error number means that the server denies permission to the object.

A process or job made an attempt to access an object in a way that the object access permissions forbid.

The process or job does not have access to the specified file, directory, component, or path.

If you access a remote file through NFS, the client will not reflect any file permissions at the client until local data updates occur. There are several options to the Add Mounted File System (ADDMFS) and MOUNT commands that determine the time between refreshes of local data. Access to a remote file may also fail due to different UID or GID mapping on the local and remote systems.

API Considerations

When using APIs that either create or remove files, there are considerations that deal with the throughput of the NFS server. If a server is busy, a client making requests can timeout and send the same requests multiple times to the server. Because of this, clients may receive either the EEXIST or ENOENT error conditions when they create or remove files using APIs. These error conditions may appear even though the operation completely successfully.

User Datagram Protocol (UDP) Considerations

To create or remove a file, the client sends a request to the server. If the server is busy, the client may timeout a number of times before the server completes the original request. The underlying UDP support of the NFS protocol may incorrectly handle these multiple requests.

UDP does not guarantee the *delivery* or *order* of data returned to clients. A client may receive any one of the following return codes for a successful operation:

1. Return code=0 (RC=0). The operation is completed successfully.
2. EEXIST. The operation is completed successfully. This error condition was returned to the client because the return code of 0 (RC=0) was either lost or received out of order.
3. ENOENT. The operation is completed successfully. This error condition was returned to the client because the return code of 0 (RC=0) was either lost or received out of order.

Client Timeout Solution

To reduce the confusion of receiving unexpected return codes and error conditions for file creation and removal, users can perform the following tasks:

1. Increase the period of client timeout so that the client will timeout less often when making server requests. This will cause fewer repeats of the same request to be transmitted to the server.
2. Increase server throughput in the following ways:
   - Start more NFS server daemons (NFSD) to handle client requests
   - Decrease the load of requests on the server
   - Decrease the amount of clients who access and make requests of the server

Network File System Differences

The following APIs have an updated set of usage notes for when users work with the Network File System:

- open()
- create()
- mkdir()
- access()
- fstat()
- lseek()
- lstat()
- stat()
- read()
- write()
- fcntl()

In general, users of these APIs should remember that local access to remote files through the Network File System may produce unexpected results. Conditions at the server dictate the properties of client access to remote files. Creation of a file or directory may fail if permissions and other attributes that are stored locally are more restrictive than those at the server. A later attempt to create a file can succeed when you refresh the local data or when you remount the file system. Several options on the Add Mounted File System (ADDMFS) command determine the time between refreshes of local data.

Once a file or directory is open, subsequent requests to perform operations on a file or directory can fail. This is because attributes are checked at the server on each request. When permissions on the object are more restrictive at the server, your operations on an open file descriptor will fail when NFS receives updates. When the server unlinks the object or makes it unavailable, then your operations on an open file descriptor will fail when NFS receives updates. The local Network File System also affects operations that retrieve file or directory attributes. Recent changes at the server may not be available at your client yet, and the server may return old values for your operations.

For more information on the usage notes for Integrated file system APIs, see the API topic in the iSeries Information Center.

open(), create(), and mkdir() APIs

If you try to re-create a file or directory that was recently deleted, the request may fail. This is because the local data that NFS stores still has a record of the existence of the file or directory. File or directory creation will succeed once you update the local data.

fcntl() API

Reading and writing to a file with the Network File System relies on byte-range locking to guarantee data integrity. To prevent data inconsistency, use the fcntl() API to get locks and release locks.

For more information on the fcntl() API, see the API topic in the iSeries Information Center.

Unchanged APIs

These APIs do not take a file descriptor, directory pointer, or path name, and are not creation functions. Therefore, NFS does not return the EACCES and ESTALE error numbers for the following APIs. All other APIs may receive the EACCESS and ESTALE error numbers.

- getegid()
- geteuid()
- getgid()
- getgrgid()
- getgrnam()
- getgroups()
- getpwnam()
- getpwuid()
- getuid()
- sysconf()
- umask()

Chapter 9. Network File System Security Considerations

You can use the Network File System to create a seamless, transparent namespace where all users have access to the right information at any given time. However, NFS also has special security considerations. These considerations deal mainly with user, group, and supplemental user identifications. This chapter discusses these concerns along with certain parameters and options of the CHGNFSEXP command.

This section describes a number of NFS security issues while explaining how to best avoid security problems and breaches while maintaining a secure namespace.

For more information about OS/400 security, see:
- The Security topic in the iSeries Information Center
- The iSeries Security Reference book in the iSeries Information Center

The Trusted Community

The trusted community is made up of only the "approved" NFS servers and clients that represent a trusted network of users. Inside this group, users export and mount file systems based on a system of individual responsibility to keep the namespace secure from outside, non-trusted users.

The other defining feature of a trusted community is that no special data encryption of any sort occurs in client/server relationships. The transmissions between the NFS clients and servers are not encoded. Only the applications running on the client will minimally encrypt and send data between client and server. This is why it is important to pay attention to how you export files from an NFS server. If the client and server transmissions are not encrypted, and you export to "the world," then anybody can access your exported file systems. For more information on exporting securely, see "Securely Exporting File Systems" on page 89.

For a detailed discussion of export options, see "Export Options" on page 90.

Network Data Encryption

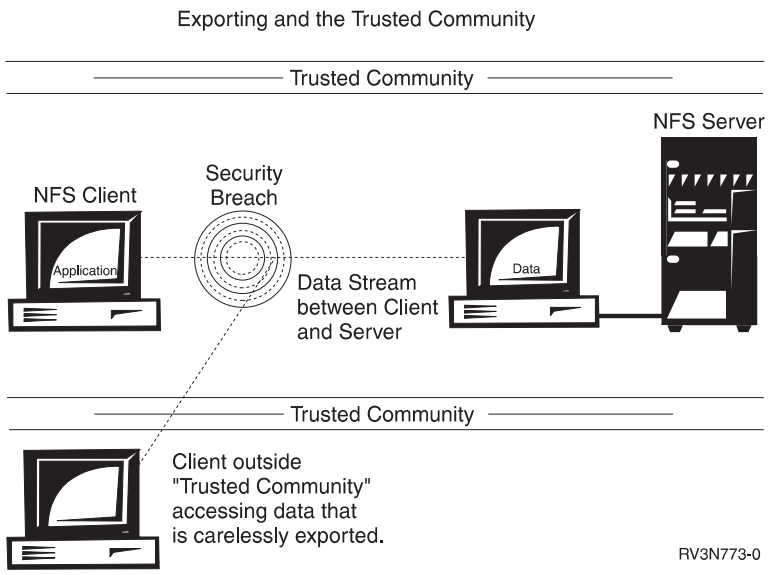

Figure 57. Client outside the trusted community causing a security breach

A client existing outside the trusted community can become aware of the community's existence. Furthermore, a malignant client can introduce a "sniff" program that can read and change data as it transfers in the client/server relationship. It accomplishes this by intercepting the data flow and altering the data on contact. The system sends unencrypted data as plain text between the client and server. Therefore, any system that is "sniffing" for such data can access it for reading by interacting with the token ring, ethernet, or other network interfaces.

There are various ways to protect the individual servers and clients in the community from outside intruders. There are also methods of protecting data as it is being sent from client to server and back again:

1. Users can code data with encryption engines, if they are located on both the client and the server. These engines code information as it leaves the "source" client or server and then decode the information when it reaches the "target" client or server.
2. The identities of users can be authenticated using the following methods:
   a. AUTH_UNIX. Authorization to objects is controlled by user identification (UID) only. There is no encryption whatsoever. iSeries automatically performs this type of object authorization.
   b. AUTH_DES. This is the Data Encryption Standard (DES). Using this type of encryption will protect data.
   c. AUTH_KERBEROS. This type of encryption protects the user through a third party administrator who administers authority tokens that are based on a trusted token manager. Kerberos security can enforce the trusted community. Kerberos is the authentication protocol used to implement private key authorization. Kerberos relies on complete and total authentication of users and authorities within the trusted community, allowing no one else access to data.

User Authorities

As users log on to NFS clients and servers, the user authority of each user dictates what they can and cannot do. User authorities are assigned to users by administrators, and usually take the form of user identifications (UIDs) for particular users, group identifications (GIDs) for groups of users, and supplemental GIDs, which list various group identifications that a user belongs to.

Properly assigning user authorities for users and systems can help you to construct a secure, efficient namespace. Users will always have access to the right data when they need it. Improperly assigning user and system authorities will lead to an insecure namespace with major internal and external security breaches.

User Identifications (UIDs)

A UID is a user identification. UIDs are made up of a number that represents the user on a particular system. The UID does not include a user's name or any other information about the user at all. UIDs are simply a way for the system to recognize users and activate their user profile for use with authorization checking. Users may have access to a various number of systems, so they could maintain a number of different UIDs and associated authorities across those systems. Users should only ever have the access authority that they need, and no more.

For example, on one iSeries, Bill might have a UID of 136. On a different iSeries, his UID could be 142. On yet another, Bill might have a UID of 2700. All of these UIDs could potentially carry with them a number of different authorities on each network system. Bill could have *USER authority on one system and *QSECOFR authority on the next, and so on. The improper administration of such varied UIDs that can lead to NFS security hazards.

Group Identifications (GIDs)

A GID is a group identification. It operates in the same fashion as a UID as a way of identifying a user's access on a given system. The GID goes one step further, however, by acting as a general authority for a group of users or machines. GIDs are not at all relative to UIDs and do not correspond with UIDs. A user profile may or may not have a GID associated with it.

GIDs, therefore, might function for a department in a company or for a set of workstations in a computer lab. GIDs can function as a second method of access to a given file or object if a particular UID does not have access authority to that file or object. A GID *never* takes away authority from the user profile associated with it; GIDs can only *add* authority. If a GID exists on the server that matches the request of a client, then the associated authority is added to the request. If the GID does not exist on the server, then the GID is ignored.

There are also supplemental GIDs, which can act as a third or fourth or *nth* method of access authority to an object. GIDs are assigned to users based on what groups a user in question belongs to. UNIX servers and clients can use supplemental GIDs to determine the authority of a user. NFS on iSeries supports supplemental GIDs. If a user on the client system is a member of multiple groups, then the system sends the GIDs for those groups to the server. The server will use the GIDs that have existing group profiles as the list of supplemental GIDs for the mapped NFS user on the server. Because GIDs and supplemental GIDs add to a user's authority, GIDs may allow users to access objects that are ordinarily forbidden to their profiles. It is important to become aware of which users from

which groups have access to your data. GIDs can help a user from a powerful group gain unauthorized access to sensitive data.

The various IDs a user has and the attached authorities can create NFS security hazards. This is particularly crucial when dealing with the CHGNFSEXP command options for making file system available to clients. For more information regarding exporting safely, see "Securely Exporting File Systems" on page 89.

Mapping User Identifications

Whenever users successfully log onto a server, the server automatically and immediately grants the authorities for their user profiles on that server.

When users access remote server files through local client systems, their requests are sent to that remote server. The server will check the user's UID and authority with each request, due to the statelessness of NFS. As a user accesses a remote server, the request carries *only* the UID and *not* the user profile name (nor any password). The server then maps the UID to a matching authority *no matter* what user profile name it actually has. This can cause problems if UIDs from different systems match each other, yet belong to different users. See "UID Mapping Examples" for more information about improper UID mapping.

Potential User Identification Mapping Scenarios

There are four possibilities for UID mapping across a distributed network:

1. The UID of a user on a client and server map to the same user profile. There is no conflict.
2. The UID of a user exists on both the client and server, but is mapped to different profiles. This can cause security conflicts because users can be mapped to profiles with more or less authority than what is required.
3. The UID of a user exists on the client, but does not exist on the server. In this case, the export entry is checked for an entry for the ANON parameter. If a profile is found for mapping anonymous users, then the UID of the user will be mapped to this profile. If the server does not allow the mapping of anonymous users, then a user making this request will receive the EACCES error condition. See "Anonymous Users" on page 90 for more information about the ANON parameter and allowing anonymous users access to your exports.

Because of differing UID mapping across a network, users may have problems working with files on a remote system. This occurs because users do not have the same authority on the remote system as on the local system.

Administrating User Identifications

The administrator of an NFS namespace must be ready to:

1. Set up matching authorities, whenever possible, for users. This ensures that they will not become confused while crossing mount points and working on both local and remote systems. This includes properly mapping UIDs and GIDs throughout the network.
2. Create appropriate individual authorities that are tailored to both the system and the user. These authorities need not be *matching*, but they should be appropriate to both the user and the system in question. Just because users need *SECOFR authority on one system does not mean that they need that same authority on *all* machines.

UID Mapping Examples

In the TULAB, an engineering graduate student named Bill has a UID of 136 on TULAB1 and a UID of 142 on iSeries TULAB2. If Bill wants to mount or otherwise

access a file on TULAB1, his UID of 136 will be transmitted to that server. TULAB1 will then run an authority check on him as 136 before it lets him access the object he requested. Even if Bill is the administrator of TULAB2, if his TULAB1 UID maps to a profile that lacks proper authority, he will *not* be able to access the object he has requested.

It is important, therefore, to make sure that all your users have properly mapped UIDs. In this case, a user was unable to access data he felt he had authority to access, but improper UID mapping can also have worse effects.

A new undergraduate engineering student named Jennifer has the exact opposite UIDs of Bill: 142 on TULAB1 and 136 on TULAB2. When Jennifer makes a request from TULAB1 to TULAB2, her UID and associated authority is transmitted over to the server. Because she and Bill have the same UID, iSeries TULAB2 will assume that she *is* Bill. TULAB2 will allow her access to all objects and actions that Bill normally has access to.

In this case, a user with little to no authority on one server unknowingly created chaos on another server simply because of poor UID mapping. It is of utmost importance to pay attention to the big picture of the namespace when assigning UIDs and their associated authorities. Give users access only to what they will need and make sure that none of the UIDs overlap. The result will be a much more secure namespace.

For example, Chris Admin can correct this situation by making sure that Bill has a UID of 136 on both systems. He can also ensure that Jennifer has a UID of 142 on both machines. Furthermore, Chris Admin can erase the UID of 142 on a system if Jennifer does not need access and, therefore, a user profile there.

In the above example, a user unwittingly assumed the UID of another user. Sometimes, however, users will knowingly impersonate the UIDs of other users. Users might attempt this to gain excess authorities and permissions to file systems or objects that were previously out of reach or to cause mischief. Either way, a smart system administrator can stop this from happening.

For example, on TULAB1 and TULAB2, there exists a user named Ray. Ray has a UID of 2700 on TULAB1, where he holds *SECADM authority. He also has a UID of 150 on TULAB2, where he is a regular user.

Another user named Joe also has regular user access to TULAB2, where his UID is 170. A problem can exist if Ray accesses Joe's user profile and discovers Joe's UID. Ray can then set up a FOOL=170 UID on TULAB1 and make requests to TULAB2. When the server checks Ray's FOOL UID, it will assume that FOOL is the same as Joe. TULAB1 will then grant Ray access to Joe's home directory and any objects that he owns. Ray then has the authority to completely delete Joe's home directory, remove Joe's access to the system, and generally perform any other act that Joe was capable of.

There are other ways to tap into a system of users and the objects owned by them. The administrator of a client can deliberately impersonate a remote server UID.

For example, the administrator of a client can log on and access the UID of a user, Mary on TULAB2, who accesses the client. If the UID of Mary is 123, then the client administrator can assume this UID and, in effect, *become* Mary on both systems.

There exists on TULAB2 a file named /home/mary/index.html. Before any users perform actions on this file, they must have its file handle. Users with proper authorities can get this handle by making a request to the parent directory. The parent directory of /home/mary/index.html is /home/mary. A file handle is a unique identifier of an object that describes the object without ever changing. A file handle will last through multiple IPLs, crashes, and so on.

If the client administrator successfully impersonates Mary with a UID of 123, then the administrator can gain access to file handles. The client administrator would then have the ability to read, write, change, and delete Mary's files.

The solution in this case is for namespace administrators to follow the rules of proper and appropriate UID mapping across the namespace. Administrators also need to pay attention to how they export file systems. The trusted community should not be opened up to outside clients or anonymous users who request access.

Proper UID Mapping

An administrator needs to properly map UIDs to provide a secure namespace where users have access to the appropriate information on the appropriate servers. There is a step-by-step process for doing this one user and server at a time.

For example, a user named Joan has a UID of 27 on TULAB1 and a UID of 14 on TULAB2. Chris Admin feels that these UIDS should be the same across the two systems to provide matching profiles that he has set up. If Joan owns any objects within the integrated file system, then Chris Admin cannot change her UID on TULAB2. He performs this series of events:

1. Chris Admin chooses one UID number for Joan's universal Network File System UID. This number will be Joan's UID across the entire network namespace. In this case, Chris decodes that JOAN=27.
2. He creates the temporary UID of TEMP=27 on TULAB2. If this UID is not already in use and processes correctly, then Chris Admin continues with the process.
3. He uses the CHGUSRPRF (Change User Profile) command to change the UID of TEMP to any unused number (12345).
4. Chris Admin then uses the DLTUSRPRF (Delete User Profile) command to delete all of Joan's directories and objects from TULAB2. Before he completes this command, however, he uses an option to transfer ownership of all these objects to TEMP.
5. He then uses the CRTUSRPRF (Create User Profile) command to create JOAN=27 on TULAB2.
6. Chris Admin uses the DLTUSRPRF command once again to delete the TEMP user profile and restore ownership of all objects back to Joan, whose universal NFS UID is now 27.

However, instead of changing the UID and user profile for each user on each system, administrators can use the QSYCHGID API. This new API can be called from iSeries command lines, C programs, COBOL programs, and through other interfaces as well. This function can change the UIDs and GIDs of both system-provided profiles and regular user profiles. To use this command, an administrator must include the following entries:

1. A ten-character profile name. This is the name of the user or system profile that is to be changed. An example of such a profile name would be BILL or QUSER.

2. A UID value. This is the new value for the UID of the user or system profile that you specified. A value of '-1' indicates that the UID will not change.
3. A GID value. This is the new value for the GID of the user or system profile that you specified. A value of '-1' indicates that the GID will not change. A value of '0' indicates that the GID will be removed from the user or system profile.
4. A pointer to an error code structure. This pointer cannot be provided from a command line, although it works through a programming interface. If a value of '0' is specified for this parameter, the user will be sent exceptions if the command fails to function.

To operate QSYCHGID, an administrator must first end all jobs that are currently running under the system or user profile name. If this is a system-provided profile, iSeries may require an entrance into "restricted state," meaning that all interactive, batch, daemon, and request jobs are ended. You can establish a "restricted state" with the ENDSBS command. All that should be left operating is the system console, the console that is directly attached to iSeries.

QSYCHGID will update a user or system profile with the appropriate UID value, as well as all objects that a user or system owns. If the objects that are owned by a user or system fail to be updated, then you should use the Reclaim Storage (RCLSTG) command to reclaim any possible lost objects.

Note: QSYCHGID will always fail if a UID number is chosen that is already allocated to an existing user or system profile. No UID value can belong to more than a single profile.

Note: UIDs and GIDs are **not** mutually exclusive. That is, a UID value of 500 and a GID value of 500 do not signify the same profile.

Securely Exporting File Systems

When namespace administrators export file systems, these are the rules they **should** consider:

1. Use an exact list of clients who have access to mount and otherwise work with the file systems that are exported. This will keep sensitive information out of the reach of clients that are not listed with the ACCESS option of the OPTIONS parameter of the CHGNFSEXP command.
2. Use QNFSANON extremely carefully when allowing anonymous UIDs access to exported file systems. When coupled with the ROOT parameter, it can give unknown (anonymous) users access to your namespace.
3. Always exclude access to the `/etc/exports` file. Do **not** export this file. A user may find a way to access this file and make data available to other users that would otherwise be safely guarded.

When administrators export file systems, they should **never** do the following:

1. Administrators should *never* export the "root" (/) Directory. Remember that whenever you export a file system, you also export all of the directories and objects "downstream" of the path. Should the "root" (/) directory become exported, all the other directories and objects downstream of "root" (/) will become exported as well. Export only what is needed by clients.
2. Administrators should *never* export any file system to "the world," allowing universal client access to information. Instead, administrators should tailor each exported file system to the needs of each client. Opening up the server to

access from “the world” allows for unknown users to mount and change your files at will. Export only to specific clients.

See “Exporting to ″The World″” for more information about limiting your exports.

3. Administrators should *very cautiously* give QNFSANON universal access to any file system whatsoever. Doing so allows unknown (anonymous) users to mount and change file systems. Export only to clients who need file systems.

   See “Anonymous Users” for more information about limiting access to anonymous users.

Export Options

There are two major ways to export a file *insecurely* to the network namespace:

1. Administrators can give anonymous users access to exported file systems by allowing unknown users to read and make changes to data.
2. Administrators can export to “the world,” meaning that all those users both within and outside of the trusted community can access the exported data.

It is not necessary to distribute file systems to unknown users or systems not in the trusted community. The administrator of an NFS server should only ever export the proper data for the proper clients at any given time.

Anonymous Users

To prohibit anonymous users from gaining access to exported file systems, it is important to completely understand the ANON option of the CHGNFSEXP command. You can specify that anonymous or unknown UIDs be mapped to the QNFSANON UID, giving those users a level of access on iSeries.

To prevent this from occurring, an administrator can specify `ANON=-1` to prohibit anonymous users from mapping to QNFSANON on the system. Remember that the default option value is QNFSANON, which translates to giving anonymous users *USER authority. This will allow anonymous users to read files, but not write to them. Administrators can also specify a UID with a different default authority with this option.

When users are mapped to QNFSANON, any objects they create belong to that profile, and *not* to their user profile. For example, a user named Cayce has a UID of 123 and mounts a server file system on a client. Cayce then creates a file. The owner of this file is QNFSANON, *not* Cayce. A display of the permissions reveals the read, write, and execute authorities belong only to the owning profile, QNFSANON. Cayce has no access to the file he just created because he is not the owner.

The solution to this problem is to follow one of these procedures:

- Create the file with open permissions for “the world”
- Change the file permissions for “the world” while still mapped to QNFSANON

Exporting to ″The World″

Instead of making exported data accessible to everyone, an administrator can employ the technique of specifying selective clients. Administrators can use the ACCESS option of the CHGNFSEXP command to use this technique. This option will also accept particular UIDs, GIDs, and supplemental GIDs. Using the ACCESS option limits access to exported files to only the specified clients. The server will ignore access requests from all other clients. The file `/etc/netgroup` allows you to create names for groups of clients. These netgroup names can be used on the

ACCESS option of the CHGNFSEXP command. For more information on the use of netgroups in exporting, see “How Do I Export File Systems?” on page 30 and “/etc/netgroup File” on page 98.

The only method of limiting access to NFS exports is to use the ACCESS= option the export command. Even if you add values to or change the RW= and ROOT= options, the export command will still provide read-only access to “the world.” The RW= option gives only the listed clients read/write access to the mount. Using the ROOT= option disallows root access for any hosts that are not listed. Using the ACCESS= option makes the export accessible to only the clients that are listed.

If users specify no options with the CHGNFSEXP command, the default is to export file systems to “the world,” allowing all clients access to data. This can cause a major breach of security because information that may be sensitive will be passed outside of the NFS namespace trusted community. Administrators should only ever export data to those who need it.

Root User Mappings

Lori has a UID of 165 and *ALLOBJ authority on a remote TULAB2. She also has a UID of 165 on a local XHOST1, a UNIX client. If Chris Admin has correctly exported the file systems Lori requests access to on which she has *ALLOBJ authority, then Lori’s requests for access will fail. However, if Lori is on the list of ROOT= clients, her requests will not fail.

UNIX servers and iSeries servers map UIDs differently. A UNIX server will treat a user that comes in with a UID of 0 as a root user. If the client originating the request is not on the ROOT= list in the export options, the user is mapped to the anonymous user. The iSeries’ definition of a root user is broader. If the incoming UID maps to any user profile with *ALLOBJ authority on the server, that user is mapped to the anonymous user profile. This is only if the request came from a client that is not on the ROOT= list. In this example, even though Lori is not the root user on her UNIX client, she is still be mapped to the anonymous profile on TULAB2. This is because the profile with UID 165 has *ALLOBJ special authority. The iSeries considers her a root user. This ensures greater security for exported file systems by not allowing remote users to assume this special authority unless specifically specified on the export request.

Appendix A. Summary of Common Commands

Table 1. CL Commands Used in Network File System Applications

Command	Description
ADDMFS	Add Mounted File System. Places exported, remote server file systems over local client directories.
CHGNFSEXP	Change Network File System Export. Adds or removes directory trees to the export table of file systems that are exported to NFS clients.
CRTUDFS	Create UDFS. Creates a User-Defined File System.
DLTUDFS	Delete UDFS. Deletes a User-Defined File System.
DSPMFSINF	Display Mounted File System Information. Displays information about a mounted file system.
DSPUDFS	Display UDFS. Displays information about a User-Defined File System.
EXPORTFS	Change Network File System Export. Adds or removes directory trees to the export table of file systems that are exported to NFS clients.
ENDNFSSVR	End Network File System Server. Ends one or all of the NFS daemons on the server and the client.
MOUNT	Add Mounted File System. Places exported, remote server file systems over local client directories.
RMVMFS	Remove Mounted File System. Removes exported, remote server file systems from the local client namespace.
RLSIFSLCK	Release Integrated File System Locks. Releases all NFS byte-range locks held by a client or on an object.
STATFS	Display Mounted File System Information. Displays information about a mounted file system. This command is an alias for the DSPMFSINF command.
STRNFSSVR	Start Network File System Server. Starts one or all of the NFS daemons on the server and client.
UNMOUNT	Remove Mounted File System. Removes exported, remote server file systems from the local client namespace.

Appendix B. Understanding the /etc Files

A directory named /etc exists within the integrated file system namespace. This directory contains important system files that users should never write to or change unless they are experienced NFS administrators. NFS uses these files to perform specific system functions. The following files are located in the /etc directory:

- /etc/exports
- /etc/netgroup
- /etc/rpcbtab
- /stc/statd

Whenever you use the export command, the command searches for the /etc/exports and /etc/netgroup files. If they do not exist, the system creates these files by using characters from the ASCII default CCSID of the system. When the system creates these files, they will contain introductory help information. If you update these files at any time, you must use the export command (with the '-A' option) to make your exports current.

Editing files within the /etc directory

The files stored within the /etc directory are stream files. You can edit them in the following ways:

1. Using the Edit File (EDTF) command
2. Using a PC-based editor
3. Using a UNIX editor through NFS

Edit stream files using the Edit File (EDTF) command

The Edit File (EDTF) command allows you to edit stream files or database files. By default, this command is not part of the base operating system. A Program Temporary Fix (PTF) provides the command. The PTF in V3R7 is SF38832. Refer to the cover letter of the PTF for instructions on how to install the commands it contains. Once you have installed the PTF, you can edit the desired file by specifying the absolute or relative path name.

The difference between the absolute and the relative path name is the "/" at the beginning of the path name. The "/" as the first character signifies an absolute path name. Absolute path names start from the root directory every time. Relative path names start from the current directory.

Absolute Path Name

```
EDTF STMF('/etc/exports')
```

This command edits the stream file exports within the directory /etc.

```
EDTF FILE(PGMLIB/ACCOUNT) MBR(ACC1)
```

This command edits the file member ACC1 within the physical file PGMLIB/ACCOUNT.

Relative Path Name

```
EDTF STMF('exports')
```

This command edits the file `exports` within the current directory.

In the EDTF utility, you find all the appropriate line commands by pressing the F1 (Help) Function Key.

Note: You can also use option 2 on the WRKLNK command to edit the file.

Edit stream files using a PC-based editor

The second way to edit the netgroup file is using a PC-based editor. Use either iSeries Navigator or File Transfer Protocol (FTP) to access the `/etc` directory on the iSeries. If you are using iSeries Navigator, you must have access to the `/etc` directory.

To use FTP to edit a stream file, perform the following steps:

1. Transfer the file to your PC using FTP.
2. Perform your desired changes using the editor of your choice.
3. Transfer the file back to the `/etc` directory using FTP.

Edit stream files using a UNIX editor through NFS

A third way to edit stream files on the iSeries server is using a UNIX editor through NFS. In order to do this, perform the following steps:

1. Export the `/etc` directory on the NFS server using the EXPORTFS command.
2. Mount the `/etc/` directory to your UNIX system by using the MOUNT command.
3. Change into the `/etc` directory by using the Change Directory (CD) command.
4. Edit the desired file by using the VI editor or an editor of your choice.

/etc/exports File

The`/etc/exports file` contains the default exports for the NFS server. The file is processed whenever the STRNFSSVR command is used with the *ALL parameter. When created by the system, the file is written in ASCII characters.

Formatting Entries in the /etc/exports File

There are several restrictions on the format of path names in the `/etc/exports` file:

1. All entries are ended by the end of a line.
2. The continuation character '\' can be used to extend one entry across multiple lines when it is the last character in the line.
3. All path names must be absolute, beginning at "root" (/).
4. The first character must be a slash (/).
5. There can be no spaces within the path name.
6. Elements of a path must be separated by the '/' character.
7. Each path element cannot be longer than 255 characters.
8. No hexadecimal characters are allowed.
9. No control-characters are allowed (this includes any characters below ASCII x020).
10. No tabs or line feeds can be used in the path name.

11. All characters following the pound sign '#' are considered comments until the end of the line. The only exception to this rule is the HOSTOPT parameter, which uses the '#' character as a starting point for each HOSTOPT entry.

Formatting the HOSTOPT (Host Options) Parameter

In the /etc/exports file, you can provide additional information for each export entry related to the remote NFS clients who will access your entry. For each remote NFS client, you can specify the following items:

```
/path_name #HOSTOPT HostName=client1,
                    PathNameCCSID=NNN,
                    DataFileCCSID=NNN,
                    NoWaitForWrites
```

HostName

The name of the remote NFS client for which you are specifying additional export options. You should specify this client by name in one of the lists as a host that has access to this export entry. You can also use netgroups for this entry. You can list multiple clients and netgroups for every entry, each with its own list of specific export options.

For more information about the HostName option on the HOSTOPT parameter, see "CHGNFSEXP/EXPORTFS Display" on page 33.

PathNameCCSID

The path name CCSID is used for the path name components of the files exported to and mounted on the specified NFS client or netgroup. For any clients or netgroups not specified on the HostName parameter, the default network path name CCSID (ASCII) is used.

DataFileCCSID

The data file CCSID is used for data of the files exported to and mounted on the specified NFS client or netgroup. For any hosts not specified on the HostName parameter, the default network data CCSID (binary, no conversion) is used.

NoWaitForWrites

The NoWaitForWrites option specifies whether write requests are handled synchronously or asynchronously for the NFS client or netgroup name. The absence of this option means that data will be written to disk immediately (synchronously). If this flag is used, then the data is not guaranteed to be written to disk immediately and can be used to improve performance (asynchronously).

Additional Format Rules for the HOSTOPT (Host Options) Parameter

1. The options must start with #HOSTOPT. There must only be one #HOSTOPT although you could have many host names.
2. Additional host names and their options must be separated by a colon.
3. Options are case-insensitive.
4. Options that are not specified will be processed as the defaults described in "Formatting the HOSTOPT (Host Options) Parameter".

Examples of Formatting /etc/exports with HOSTOPT Parameter

Example 1: Exporting to a host and specifying all options.

```
/home/joe access=sammy         \
#HOSTOPT HostName=sammy,       \
         PathNameCCSID=367, \
         DataFileCCSID=850, \
         NoWaitForWrites
```

A user exports /home/joe to host sammy, specifying CCSIDs for both path names and data files. Data written is not guaranteed to be written immediately to disk (asynchronously). Note the use of the continuation character '\' to break the long entry into more readable lines.

Example 2: Exporting with options to multiple clients and netgroups.

```
/home/selfe access=sammy:rainfall:dept-OA5 \
#HOSTOPT HostName=sammy,                   \
         PathNameCCSID=367:             \
         hostname=rainfall,               \
         NoWaitForWrites
```

A user exports /home/selfe to host sammy, rainfall, and the netgroup dept-OA5. For the host sammy, a CCSID for path names has been specified. For the host rainfall, data is not guaranteed to be written immediately (asynchronously) to disk.

Example 3: Exporting a directory within QSYS.LIB with options.

```
/qsys.lib/jon.lib access=dept-OA5 \
#HOSTOPT HostName=wheatley,       \
DataFileCCSID=367
```

A user exports /qsys.lib/jon.lib to the netgroup dept-OA5. A CCSID is specified for data files. This causes the data to be converted to this CCSID before being sent to the client that is requesting the data.

/etc/netgroup File

The /etc/netgroup file contains definitions for netgroups. A netgroup is a mechanism used to associate a list of systems with one name. For example:

```
department-OA5 (larrys-box,,),(safe-house,,)
my-hosts (rainfall,,),(sammy,,),(department-OA5,,)
```

The above example defines the netgroup my-hosts, which is a list of the following: rainfall, sammy, larrys-box, and safe-house. Note that larrys-box and safe-house were included because department-OA5 is itself a netgroup. Netgroups can be nested within other netgroups.

Traditionally, netgroups are defined as:

netgroup-name
(host-name,user-name,domain-name)

host-name
The name of any host from the /etc/hosts file. iSeries does not support the /etc/hosts file as a separate integrated file system file. The /etc/hosts is built in to iSeries server TCP/IP support.

user-name
The name of any user from the /etc/passwd file. iSeries does not support the /etc/passwd file concept.

domain-name
The name of any domain.

On iSeries, the following format rules apply to netgroups:

1. All entries are ended by the end of a line.
2. The continuation character '\' can be used to extend one entry across multiple lines when it is the last character in the line.
3. All characters following the pound sign '#' are considered comments until the end of the line.
4. In the triple (host-name,user-name,domain-name) only the host-name is recognized. Any value supplied for user-name or domain-name will be ignored.
5. Commas ',' should be used to separate elements in a list.
6. A blank host-name will not be considered as a wild-card. There is no wild-card support.
7. Blanks are allowed anywhere except within a netgroup-name or a host-name.
8. Imbedded hexadecimal characters are not allowed.
9. No control-characters are allowed (this includes any characters below ASCII x020).
10. Entries are not case-sensitive.
11. Triplets (x,y,z) (a,b,c) may be separated by spaces or commas only.

/etc/rpcbtab File

The `/etc/rpcbtab` file is used by the RPC binder (port mapper) daemon. Information for all daemons is stored in this file, which is used to register daemons upon recovery of an RPC binder (port mapper) daemon crash.

Note: It is recommended that users never write to or change the `/etc/rpcbtab` file.

/etc/statd File

The `/etc/statd` file is used exclusively by the server Network Status Monitor daemon. This file stores information about any clients that hold byte-range locks on open server files. When a user starts the NSM daemon after a crash, the NSM daemon contacts all the clients listed in this file so that the clients can release any locks they hold on the server.

Note: It is recommended that users never write to or change the `/etc/statd` file.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator
3605 Highway 52 N
Rochester, MN 55901-7829
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Code disclaimer information

This document contains programming examples.

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Programming Interface Information

This publication is intended to help you to use OS/400 Network File System Support to construct a distributed network. This publication documents General-Use Programming Interface and Associated Guidance Information provided by OS/400 Network File System Support.

General-Use programming interfaces allow the customer to write programs that obtain the services of OS/400 Network File System Support.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

Application System/400
AS/400
e (logo)
IBM
iSeries
Operating System/400
OS/400
400

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

PC Direct is a trademark of Ziff Communications Company in the United States, other countries, or both and is used by IBM Corporation under license.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be the trademarks or service marks of others.

Bibliography

This bibliography lists publications that contain background information or more details for information that this book discusses.

The list below gives the full title and order number of each book. OS/400 Network File System Support may use a shortened version of the title when referring to these books.

- CL topic in the iSeries Information Center (formerly SC41-4722)

  These articles provide a description of iSeries Control Language (CL) and commands. Each command description includes a syntax diagram, parameters, default values, keywords, and an example.
- Exploring the NFS on AS/400 (SG24–2158)

  This redbook explores the Network File System on the iSeries server. It helps you use the features of the NFS to share data across systems. It further explains topics that include security considerations, National Language Support, migrating from FSS/400 to NFS, and a few common problems you might come across while using the NFS and how to solve them.
- NetWare on iSeries topic in the iSeries Information Center (formerly SC41-4124)

  These articles describe how to install and administer Novell NetWare on an Integrated xSeries Server for iSeries that is attached to an iSeries server. They also describe how to install and administer the Enhanced NetWare function on the iSeries server.
- Integrated File System topic in the iSeries Information Center (formerly SC41-4711)

  These articles describe the associated concepts and terminology of the integrated file system for the iSeries server. They also discuss interfaces, CL commands, and APIs, as well as the specific characteristics of each iSeries file system.
- Optical Support (formerly SC41-4310)

  This book serves as a user's guide and reference for IBM Optical Support on OS/400. The information in this book can help the user understand optical library dataserver concepts for use in administering an optical library dataserver.
- Basic system security and planning topic in the iSeries Information Center (formerly SC41-4301)

  These articles provide general information about OS/400 security and authorities.
- Security Reference (formerly SC41-4302)

  This book provides detailed technical information about OS/400 security.
- API topic in the iSeries Information Center (formerly SC41-4801)

  These articles provide a description of each OS/400 API, including the integrated file system APIs.
- Work management topic in the iSeries Information Center

  These articles describe work and job management concepts.

Index

Special Characters

A

B

C

D

E

F

G

K

M

N

P

Q

R

S

T

U

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC41-5714-02