Data General
Data General Corporation, Westboro, Massachusetts 01580

Customer Documentation

NetWare® for AViiON® Systems: C Interface Programmer's Guide

069–000566–00

AViiON®
PRODUCT LINE

NetWare® for AViiON® Systems: C Interface Programmer's Guide

069–000566–00

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085–series) supplied with the software.

Rev. 00, January 1992
Licensed Material — Property of the copyright holder(s)
Ordering No. 069–000566

Notice

NetWare® for AViiON® Systems:
C Interface Programmer's Guide
069–000566–00

Revision History:

Original Release – January 1992

Effective with:

NetWare® for AViiON® Systems,
Revision 1.30

Preface

This manual provides background information about the Applications Programming Interface (API) for NetWare® for AViiON® Systems file servers. It explains the methods of operation and the services that are available to application programmers. *NetWare® for AViiON® Systems: C Interface Reference Guide* provides detailed information about the specific function calls that control those services.

This manual is a descendant of the *Application Programmer's Guide to NetWare®*, published in February 1986. While the earlier manual was oriented toward BASIC programmers who write application programs, this manual has a broader scope. The NetWare for AViiON Systems APIs are designed for application developers who want to write DG/UX™ applications independent of the NetWare shell. Applications written with these APIs are compatible with NetWare 286, NetWare 386 and NetWare for AViiON Systems servers, unless otherwise noted.

NetWare for AViiON Systems is compatible with NetWare version 3.0, unless otherwise noted. All references to NetWare version 3.0 in this manual apply to NetWare for AViiON Systems. The release notice accompanying your shipment provides the most current information about exceptions to this compatibility.

IMPORTANT: Unless otherwise noted, NetWare for AViiON Systems servers cannot service API programs running in a NetWare network.

Organization of this manual

Chapter 1 Accounting Services

This chapter explains how to use the Accounting Services that allow servers to charge clients for services.

Chapter 2 Bindery Services

This chapter explains how to control the Bindery Services that regulate access to the file server.

Chapter 3 Connection Services

This chapter explains how to establish and destroy logical connections to the file server and control the return of status information about those connections.

Chapter 4 File Services

This chapter explains how to regulate the File Services that enable applications to manipulate files, directories, volumes, trustees, and their associated information.

Chapter 5 Path Services

This chapter explains how to set up the Path Services to allocate directory handles and return information about directory paths.

Chapter 6 Queue Services

This chapter explains how to set up the Queue Management Services to control the flow of jobs and services on the network.

Chapter 7 Server Platform Services

This chapter explains how to write function calls that report file server information and disk usage.

Chapter 8 Synchronization Services

This chapter explains how to use the Synchronization Services to coordinate access to network files and other resources.

Chapter 9 Transaction Tracking Services

This chapter explains how to use Transaction Tracking System attributes to ensure file integrity of selected files.

Related Documents

You received a comprehensive set of documents with your NetWare for AViiON Systems release package. The manuals listed below are included in that set and contain information that augments the text of this manual.

NetWare® for AViiON® Systems: C Interface Reference Guide (069–000567)

This manual provides a comprehensive set of command instructions and API program calls for all available NetWare API services and functions, and a complete cross reference to the previously-released DOS API library. It is written specifically for applications programmers.

NetWare® for AViiON® Systems: Concepts (069–000483)

This manual provides an alphabetically-arranged glossary of NetWare terminology. It is written for all levels of NetWare users, but it will be particularly useful to supervisors who are performing their first installation of the NetWare for AViiON Systems product.

NetWare®for AViiON® Systems: Installation (069-000488)

This manual provides detailed instructions for planning a NetWare network, installing NetWare for AViiON Systems on an AViiON computer, configuring client workstations, and setting up user accounts. It is written for the network supervisor.

NetWare® for AViiON® Systems: System Administration (069–000487)

This manual provides a reference to the SCONSOLE and HYBRID utilities and the NetWare for AViiON Systems printing services and utilities. It is written primarily for network supervisors who will use SCONSOLE and HYBRID to administer the AViiON file server and set up DG/UX printers using the NetWare printing utilities.

NetWare® for AViiON® Systems: User Book (069–000486)

This manual provides a general overview of NetWare. It is written for first-time users who are unfamiliar with networks.

NetWare® for AViiON® Systems: Utilities (069–000484)

This manual provides an alphabetically-arranged reference for NetWare command line and menu utilities. It is written for all levels of NetWare users.

Reader, please note

In all examples within the text, we use

```
This typeface to show system prompts and responses.
```

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please contact your local Data General sales representative.

Telephone assistance

If you are unable to solve a problem with your system, free telephone assistance is available with your warranty and with most Data General service options. If you are within the United States or Canada, contact the Data General Customer Support Center (CSC) by calling 1–800–DG–HELPS. Lines are open from 8:00 a.m. to 5:00 p.m., your time, Monday through Friday. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining our user's group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive *FOCUS* monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual *Member Directory*, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1–800–253–3902 or 1–508–443–3330.

End of Preface

Contents

Chapter 1
Accounting Services

Accounting Services is an Applications Programming Interface (API) that allows a server to charge for the use of its services. For example, a database server can charge for the number of records viewed, the number of requests serviced, or the amount of connect time. A print server can charge for the number of pages printed.

The file server supervisor determines the charge for each type of service, and the file server bindery stores the list of authorized accounting servers and each client's accounting information.

Once a server is listed as an authorized accounting server and has logged in, it can submit a charge to a client's account, place a hold against a client's account, or query a client's account status. An audit trail of all charges is accumulated in an audit file. This chapter is divided into the following sections:

- Bindery Properties
- Accounting Audit File
- File Server Charges
- Data Structures

Bindery Properties

Information used by the accounting APIs is stored in three bindery properties:

- ACCOUNT_SERVERS property
- ACCOUNT_BALANCE property
- ACCOUNT_HOLDS property

These properties are discussed in detail in the following sections.

ACCOUNT_SERVERS Property

When accounting is enabled (using the NetWare® utility SYSCON), the ACCOUNT_SERVERS property is created and attached to the file server object. This property contains the bindery object ID of every server, including the file server, authorized to submit accounting records to clients. Before a server can charge for its services, the server must be added to the ACCOUNT_SERVERS property.

For example, to enable a server called PRINTSERV to charge clients for services on a file server called SAM, you must complete the following steps.

1. Install accounting on the file server SAM. Log in as supervisor and run SYSCON. Select to install accounting.

2. Use NWCreateObject to create PRINTSERV as a bindery object. Contact Novell's API Consulting Group if you need a unique object type.

3. Add PRINTSERV to the ACCOUNT_SERVERS property. Use NWAddObjectToSet.

   For the objectName parameter, use the file server's name (SAM).

   For the objectType parameter, use the file server's type (NWOT_FILE_SERVER).

   For the propertyName parameter, use ACCOUNT_SERVERS.

   For the memberName parameter, use the object name created in Step 2 (PRINTSERV).

   For the memberType parameter, use the object that you declared in Step 2 (for a print server, NWOT_PRINT_SERVER).

The ACCOUNT_SERVERS property is of type set and has read-supervisor and write-supervisor security access levels. If a file server object has no ACCOUNT_SERVERS property, servers should assume that accounting is disabled, and no charges for services are made.

ACCOUNT_BALANCE Property

When accounting is enabled, each user object is given an ACCOUNT_BALANCE property. Every user object created after accounting is enabled will also be given an ACCOUNT_BALANCE property. An object's account balance and minimum balance are stored in this property. Servers can deny services to an object if the object has no ACCOUNT_BALANCE property.

The ACCOUNT_BALANCE property is of type data and has read-object and write-supervisor security access levels. The account balance places a general limitation on a client's use of all network resources. However, Accounting Services does not provide any method for placing a ceiling on a client's use of a particular resource or service.

The account balance usually represents some monetary unit such as cents. The system administrator must ensure that all servers submit their charges using the same monetary unit. If an object's account balance has no more funds, servers should refuse further service. However, if service has already been rendered, the server can charge for it even though no funds are in the account balance. Setting the minimum balance to 80000000h (the most negative signed long integer) indicates the object has no minimum balance, in which case service should never be refused.

The ACCOUNT_BALANCE property has several property fields which are described below.

Property Fields

The fields in the first data block of the ACCOUNT_BALANCE property are listed and defined below.

Table 1-1 ACCOUNT_BALANCE Property

Offset	Field	Type
0	Balance	int32
4	Minimum Balance	int32
8	Reserved	uint8 [120]

Balance. This field contains a signed integer that indicates the account balance.

Minimum Balance. This field contains a signed integer that indicates the lowest permissible balance. Services that cause the balance to drop below this minimum are denied. The minimum can be positive (requiring the user to always maintain some funds) or negative (allowing the user to receive services after the balance has dropped below zero.)

Reserved. This field contains NetWare internal information.

ACCOUNT_HOLDS Property

The server can place a hold on a client's account before a request is serviced to ensure that the client has sufficient funds to pay for service. If the hold succeeds, this indicates the client has sufficient funds. The server can then perform the requested service, submit a charge for the service, then release the amount placed on hold.

If there are holds against a client's account, the holds are stored in the ACCOUNT_HOLDS property. If there are no holds on the account, this property is not present. The ACCOUNT_HOLDS property is dynamic, of type data, and has read-object and write-supervisor security access levels.

If a hold, along with all other holds, require more funds than exist in the client's account (client's funds have gone below minimum balance), the hold request fails. The requested service should then be refused. An attempt to hold also fails if 16 other severs have already placed holds on a client's account.

If the server does not place a hold on an account to ensure the client has sufficient funds, the server should query the account before rendering service. Then, if the client's account is empty, the request for service should be refused. If a holding server is disconnected, the hold it had on a client's account is cleared. The holding server can also explicitly clear a hold. If a server submits multiple holds, they accumulate into one hold.

The ACCOUNT_HOLDS property has several property fields which are described below.

Property Fields

The fields in the first data block are shown in Table 1-2.

Table 1-2 ACCOUNT_HOLDS Property

Offset	Field	Type
0	Holder ID 1	uint32
4	Amount 1	uint32
	. . .	
120	Holder ID 16	uint32
124	Amount 16	uint32

Holder ID. This field contains the bindery object ID of the server placing the hold. If this field is zero, the hold slot is not in use regardless of the contents of the amount field.

Amount. This field contains the amount of the hold.

Accounting Audit File

In addition to monitoring a network by charging, querying, or holding a client's account, Accounting Services offers a method for monitoring network resources by keeping an audit trail of all charges. The two APIs, NWSubmitAccountCharge and NWSubmitAccountNote and the Accounting Audit file (NET$ACCT.DAT) are the tools used to keep the Audit trail.

The audit file has normal attributes and resides in the SYS:SYSTEM directory. It can be read by utility programs and can be deleted or renamed with standard DG/UX commands.

When the two APIs are called, a detailed record of all accounting charges and activities is kept in the Accounting Audit file, NET$ACCT.DAT. For example, when a server submits a charge or a note, a Charge Record or a Note Record is added to the audit file. This section describes the structure of these two records and then describes in detail two of the records' fields: the comment type field and the comment field.

Charge and Note Records

The Charge record and Note record are distinguished by the contents of their Record Type field. When the server charges an account through the NWSubmitAccountCharge function call, a Charge record is added to the NET$ACCT.DAT audit file.

This Charge record contains information about the charge such as the object ID of the server submitting the charge, when the charge is submitted, the amount of the charge, etc.

Likewise, when the server calls the NWSubmitAccountNote function call, a Note record is added to the NET$ACCT.DAT audit file. It also contains information such as the bindery object ID of the server submitting the note, when the note is submitted, etc.

Both records also contain a Comment field.

- **Charge record.** The Comment field of the Charge record contains a binary record that holds information about the charge. For example, the file server uses the comment field of the Charge record to record the number of service units (connect time, disk I/Os, packet I/Os, etc.) from which a charge is computed.
- **Note record.** The Comment field of the Note record contains information such as whether a station is logging in or out, and when it is doing so. This field also contains the physical address of the station logging in or out.

A more detailed list and description of each record's fields are provided below.

Charge Record Structure

The Charge record structure is shown below.

Table 1-3 Charge Record Structure

Offset	Field	Type
0	Length	uint16
2	Server ID	uint32
6	Time Stamp	unit8 [6]
12	Record Type	uint8 [1]
13	Completion Code	uint8 [1]
14	Service Type	uint16
16	Client ID	uint32
20	Amount	uint32
24	Comment Type	uint16
26	Comment	char [*]

Length. This field contains the number of bytes in the record excluding Length itself.

Server ID. This field contains the object ID of the server submitting the accounting request.

Time Stamp. This field contains the date and time the server submitted the charge. The format is year (year = current year - 1900), month, day, hour, month, second, where each is a byte-sized integer.

Record Type. This field contains the record type: A Charge Record is type 1, and a Note Record is type 2.

Completion Code. This field contains the completion code of the Submit Charge request. If the completion code is SUCCESSFUL (00) or CREDIT_EXCEEDED (C2), the account balance is debited.

Service Type. This field contains the specific type of service for which the charge is made. External servers should use their object type. If a server provides several different services, and no reasonable object type equivalents exist for these services, you should contact Novell's API Consulting Group for a well-known service type. Usually, however, the server should use its object type and distinguish the type of service being charged for in the comment field of the charge record.

Client ID. This field contains the object ID of the object that is being charged for service.

Amount. This field contains the amount of the charge. Amount can be negative if the server must make a refund.

Comment Type. This field contains the type of the comment record. The comment type is used to locate the associated record descriptor in the comment record definitions file (NET$REC.DAT). That record descriptor contains the field layout and text descriptions for the comment record. (See "Comment Type Field Definitions" in this chapter for more information on comment fields.)

Comment types are administered by Novell. Contact Novell's API Consulting Group for unique comment types. Comment types greater than 8000h are reserved for experimental purposes.

Comment. The Comment field of the Charge Record contains a binary record that holds information about the charge. For example, the file server uses the comment field to record the number of service units (connect time, disk I/Os, packet I/Os, etc.) from which a charge is computed. (See "Comment Field Definitions" in this chapter for more information on comment fields.)

Note Record Structure

The Note Record structure is shown below.

Table 1-4 Note Record Structure

Offset	Field	Type
0	Length	uint16
2	Server ID	uint32
6	Time Stamp	unit8 [6]
12	Record Type	uint8 [1]
13	Reserved	uint8 [1]
14	Service Type	uint16
16	Client ID	uint32
20	Comment Type	uint16
22	Comment	char [*]

Length. This field contains the number of bytes in the record excluding Length itself.

Server ID. This field contains the object ID of the server submitting the accounting request. A Server ID of zero indicates a request by an internal server.

Time Stamp. This field contains the date and time the server submitted the note. The format is year (year = current year - 1900), month, day, hour, month, second, where each is a byte-sized integer.

Record Type. This field contains the record type: Charge Record is type 1, and Note Record is type 2.

Service Type. This field contains the specific type of service to which the note applies. External servers should use their object type. If a server provides several disparate types of services, and no reasonable object type equivalents exist for these services, you can contact Novell's API Consulting Group for a well-known service type. Usually, however, the server should use its object type and distinguish the type of service for which the note is produced in the comment field of the note record.

Client ID. This field contains the object ID to which the note applies.

Comment Type. This field contains the type of the comment record. The comment type is used to locate the associated record descriptor in the comment record definitions file (NET$REC.DAT). That record descriptor contains the field layout and text descriptions for the comment record. (See "Comment Type Field Definitions" below for more information on comment fields.)

Comment types are administered by Novell. Contact Novell's API Consulting Group for unique comment types. Comment types greater than 8000h are reserved for experimental purposes.

Comment. This field contains a binary record that holds information relating to the charge or note. For example, the file server records in this field whether a station is logging in or out, and when it is doing so. This field also contains the physical address of the station logging in or out. (See "Comment Field Definitions" in this chapter for more information on comment fields.)

Comment Type Field Definitions

The NWSubmitAccountCharge function call and the NWSubmitAccountNote function call allow a file server to record information in the audit file about a client's account activities. This information is a binary record in the Comment field of the Note Record or Charge Record. These record types are described below.

Connect Time Charge

The Comment Type is 1.

This record contains the number of minutes a station was connected to the server, the number of packets sent to the server, and the number of disk I/Os.

The fields in this comment type are listed below.

Table 1-5 Connect Time Charge

Offset	Field	Type
0	Connect Time	uint32
4	Request Count	uint32
8	Bytes Read Hi	unit16
10	Bytes Read 2	uint16
12	Bytes Read Lo	uint16
14	Bytes Written Hi	uint16
16	Bytes Written 2	uint16
18	Bytes Written Lo	uint16

The format control string:

"Connected %lu minutes; %lu requests; %04x%04x%04xh bytes read;
%04x%04x%04xh bytes written."

Disk Storage Charge

The Comment Type is 2.

This Comment contains the number of blocks owned by an account at the time of the charge and the number of days in the charge period. The fields in this comment type are listed below.

Table 1-6 Disk Storage Charge

Offset	Field	Type
0	Blocks Owned	uint32
4	Number Of Half Hours	uint32

The format control string:

"%lu disk blocks stored for %lu half-hours."

Log In Note

The Comment Type is 3.

This Comment is recorded whenever an object successfully logs in. The Comment field of the Note Record contains the physical address of the station logging in. The fields in this comment type are listed below.

Table 1-7 Log In Note

Offset	Field	Type
0	Net	int32
4	Node (high)	int32
8	Node (low)	uint16

The format control string:

"Login from address %lx:%lx%x."

Log Out Note

The Comment Type is 4.

This Comment is recorded whenever an object logs out. The Comment field of the Note Record contains the physical address of the station logging out. The fields in this comment type are listed below.

Table 1-8 Log Out Note

Offset	Field	Type
0	Net	int32
4	Node (high)	int32
8	Node (low)	uint16

The format control string:

"Logout from address %lx:%lx%x."

Account Locked Note

The Comment Type is 5.

This Comment is recorded whenever an account is locked because of too many incorrect login attempts. The Client field of the Note Record contains the object ID of the object being locked out. The Comment field of the Note Record contains the physical address of the station being locked out. The fields in this comment type are listed below.

Table 1-9 Account Locked Note

Offset	Field	Type
0	Net	int32
4	Node (high)	int32
8	Node (low)	uint16

The format control string:

"Account intruder lockout caused by address %lx:%lx%x."

Server Time Modified Note

The Comment Type is 6.

This Comment is recorded when an operator modifies the server time. The Time Stamp field contains the current time before the change was made. The Client field contains the object ID of the object which changed the time. The comment field contains the new time and date. The fields in this comment type are listed below.

Table 1-10 Server Time Modified Note

Offset	Field	Type
0	Year Since 1900	uint8 [1]
1	Month	uint8 [1]
2	Day	unit8 [1]
3	Hour	uint8 [1]
4	Minute	uint8 [1]
5	Second	uint8 [1]

The format control string:

"System time changed to 19%02d-%02d-%02d %d:%02d:%02d."

Comment Field Definitions

The NET$REC.DAT file is a companion file to the NET$ACCT.DAT Audit file. It contains the information for formatting and the control strings for displaying the binary records in the Comment fields of the Charge and Note Records. Utility programs use the information in NET$REC.DAT to display the Charge and Note records from the NET$ACCT.DAT audit file. The information in the NET$REC.DAT file is listed in Table 1-11.

Table 1-11 NET$REC.DAT File

Offset	Field	Type
0	Length	uint16
2	Comment Type	uint16
4	Data Type Count	uint8 [1]
5	Data Type Value [1]	uint8
6	Data Type Element(s) [1]	(as data type defines)
	. . .	
	Data Type Value [n]	uint8
	Data Type Element(s)	(as data type defines)

Length. This field contains the length of the record descriptor, not including the Length field itself. The length is

(Field Count * 2) + Format Length + 4

Comment Type. This field contains the type of the comment being described. This Comment Type is also a field in the Charge and Note records, used there as a reference to this record descriptor. Well known comment types can be obtained from Novell's API Consulting Group. Comment types 8000h and higher are reserved for experimental purposes.

Data Type. This field contains the data type of the record. Valid types are listed below.

Table 1-12 Data Types

Value	Element Description
1	An 8-bit value
2	A 16-bit value with the bytes stored
3	A 32-bit value with the bytes stored
4	An 8-bit value followed by an array of characters. The size of the array is specified by the length value.

NOTE: You should read and write data in hi-lo format. Then, for your particular hardware, take the appropriate action.

Data Type Count. This field contains the number of data types to follow. Data types were listed in Table 1-12, above.

File Server Charges

File servers can submit charges for connection time, requests made, blocks read, blocks written, and disk storage. The following Accounting Services bindery properties describe these charges. The first four properties (CONNECT_TIME, REQUESTS_MADE, BLOCKS_READ, BLOCKS_WRITTEN) share the same data structure. The last property, DISK_STORAGE, has a different data structure. This section describes each charge. The data structures are described later in this chapter.

CONNECT_TIME Property

This property designates the amount a file server can charge for connect time. Connect time is measured from the time the user logs in to the time the user logs out. The user's account is checked each half hour. If the account balance falls below the lowest permissible balance, the user is disconnected following 5 minute and 1 minute warnings.

REQUESTS_MADE Property

This property designates the amount a file server can charge for requests made to the file server. After logging out, the user can be charged for the total number of requests submitted to the file server since logging in. While the user is logged in, the user's account is checked each half hour. If the account balance falls below the lowest permissible balance, the user is disconnected following 5 minute and 1 minute warnings.

BLOCKS_READ Property

This property designates the amount the file server can charge for blocks read. After logging out, the user can be charged for the number of blocks (4096 bytes per block) read since logging in. While the user is logged in, the user's account is checked each half hour. If the account balance falls below the lowest permissible balance, the user is disconnected following 5 minute and 1 minute warnings.

BLOCKS_WRITTEN Property

This property designates the amount the file server can charge for blocks written. After logging out, the user can be charged for the number of blocks (4096 bytes per block) written from disk since logging in. While the user is logged in, the user's account is checked each half hour. If the account balance falls below the lowest permissible balance, the user is disconnected following 5 minute and 1 minute warnings.

DISK_STORAGE Property

This property designates the amount the file server can charge for disk storage. The charge is computed at regular intervals as follows:

- The total disk storage for each user is calculated.
- Total disk storage is multiplied by the number of half hours since the last disk storage charge was made.
- The result is multiplied by the current charge rate.

Unlike charges designated in the CONNECT_TIME, REQUESTS_MADE, BLOCKS_WRITTEN, and BLOCKS_READ properties, the list of times and the charge rates designated in the DISK_STORAGE property represent a specific time a charge is made and a specific rate for the charge.

Data Structures

The file server maintains a schedule of charges for its services in bindery property data structures. The data structure for connection time, requests made, blocks read, and blocks written is described below. The data structure for disk storage is described later in this chapter.

Connection-Time, Requests-Made, Blocks-Read, and Blocks-Written Charges

The CONNECT_TIME, REQUESTS_MADE, BLOCKS_READ, and BLOCKS_WRITTEN properties share the same data structure. This structure is shown in Table 1-13.

Table 1-13 Data Structure for Four Types of Charges

Offset	Field	Type
0	Time of Next Change	uint32
4	Current Charge Rate Multiplier	uint16
6	Current Charge Rate Divisor	uint16
8	Days Change Occurs Mask 1	uint8 [1]
9	Time Change Occurs 1	uint8 [1]
10	Charge Rate Multiplier 1	uint16
12	Charge Rate Divisor 1	uint16
	.	
	.	
122	Days Change Occurs Mask 20	uint8 [1]
123	Time Change Occurs 20	uint8 [1]
124	Charge Rate Multiplier 20	uint16
126	Charge Rate Divisor 20	uint16

Time Of Next Change. This field contains the time of the next change. Time is measured in minutes since January 1, 1985.

Current Charge Rate Multiplier and Divisor. These fields contain the Current Charge Rate Multiplier and Divisor. The charge is computed as follows:

- The unit of service (connect time, requests made, blocks read, or blocks written) is multiplied by the Current Charge Rate Multiplier.
- The resulting value is divided by the Current Charge Rate Divisor to give the charge made against the user's account. (If the Current Charge Rate Multiplier or Divisor is zero, no charge is made for the service.)

Days Change Occurs Mask. This field contains a bit mask that specifies the days of the week for which the charge rate applies. If the bit corresponding to the day of the week is set (bit 0 = Sunday, ..., bit 6 = Saturday), the charge is made during that day at the half hour specified in Time Change Occurs. If this field is zero, no changes are made.

Time Change Occurs. This field contains the half hour (12am = 0, ..., 11:30pm = 47) during which the specified charge rate takes effect. These changes in the charge rate are listed in this property structure according to increasing half hours.

Charge Rate Multiplier and Divisor. These fields contain the charge rate that takes effect at the specified time.

Disk Storage Charges

Charges for disk storage are maintained in the DISK_STORAGE property. The structure of this property is described below.

Table 1-14 DISK_STORAGE Data Structure

Offset	Field	Type
0	Time of Next Change	uint32
4	Time of Previous Charge	uint16
8	Days Charge Occurs 1	uint8 [1]
9	Time Charge Occurs 1	uint8 [1]
10	Charge Rate Multiplier 1	uint16
12	Charge Rate Divisor 1	uint16
	. . .	
122	Days Change Occurs Mask 20	uint8 [1]
123	Time Change Occurs 20	uint8 [1]
124	Charge Rate Multiplier 20	uint16
126	Charge Rate Divisor 20	uint16

Time of Next Charge. This field contains the time of the next charge. Time is measured in minutes since January 1, 1985.

Time of Previous Charge. This field contains the time of the previous charge. Time is measured in minutes since January 1, 1985.

Days Charge Occurs Mask. This field contains a bit mask that specifies the days of the week for which the charge rate applies. If the bit corresponding to the day of the week is set (bit 0 = Sunday, ..., bit 6 = Saturday), the charge is made during that day at the half hour specified in Time Change Occurs. If this field is zero, no changes are made.

Time Charge Occurs. This field contains the half hour (12am = 0, ..., 11:30pm = 47) during which the specified charge rate takes effect. These changes in the charge rate are listed in this property structure according to increasing half hours.

Charge Rate Multiplier and Divisor. These fields contain the charge rate used to calculate the disk storage charge at the specified half hour on the specified days.

The charge is computed as follows:

- The current disk storage for each user is calculated in 4K blocks.
- Current disk storage is multiplied by the number of half hours since the last disk storage charge was made.
- The result is multiplied by the charge rate multiplier and divided by the charge rate divisor to produce the final disk storage charge.

End of Chapter

Chapter 2
Bindery Services

In a local area network environment there is the need for a name service which provides a way for network resources and clients to be identified. A resource is anything that provides a service such as a file server, print server or database server. A client is the consumer of the services provided by a resource. Each NetWare® file server maintains a database of the resources and clients available on the network. This special-purpose database is called the bindery. This chapter is divided into the following sections:

- Bindery Overview
- Bindery Security
- Bindery Objects
- Bindery Properties

Bindery Overview

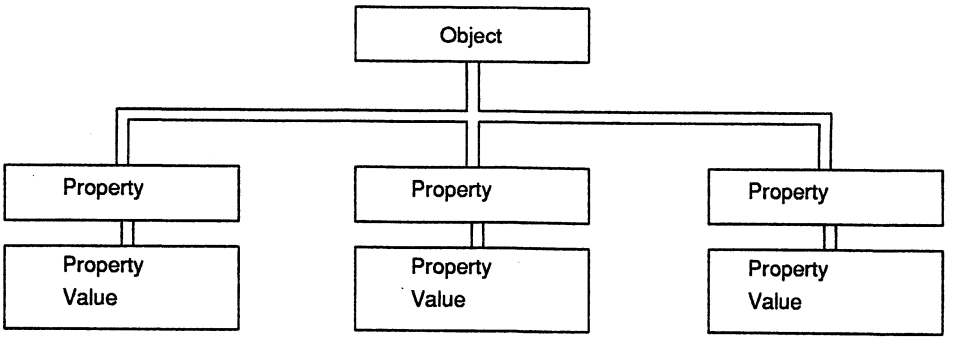
The fundamental benefit of the bindery is that it allows the design of an organized and secure operating environment. The bindery provides the foundation upon which NetWare's client security and server advertising are built. The bindery contains information on each client and is the basis upon which NetWare's client security mechanisms are built, including client password protection, client accounting and client restrictions.

The bindery also contains information about each resource on the network. For example, resources which advertise their services have their name and internetwork address stored in every file server's bindery in the internetwork. Therefore, the bindery can also be used as a resource directory where clients can extract a listing of all resources available on the network.

The bindery is not designed to be used exclusively by the NetWare operating system, but is a flexible database with an extensive programming interface that can also be used by third-party applications. Applications can access the bindery to extract information about network users and resources. The bindery can also be used to store application specific information. For example, an application's user and configuration information could be stored in the bindery.

Bindery Components

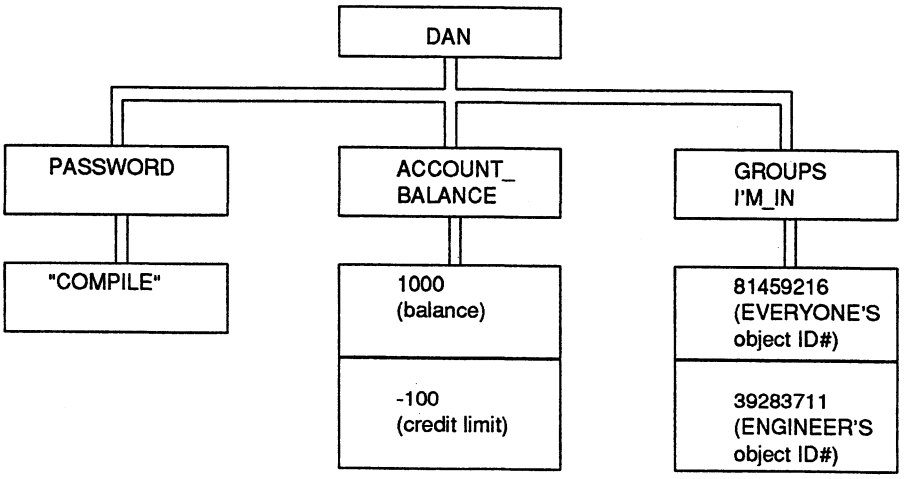
The bindery is comprised primarily of components called objects and properties. An object can be a user, user group, file server, print server, or any other logical or physical entity on the network that has been given a name. Each object has associated with it a set of characteristics called properties, as represented in the figure below.



Each object can have multiple properties associated with it. Each property may have an attached value. The property's value contains the actual data that is associated with a property.

A user's set of properties may include a password, an account balance, and a list of groups the user is a member of whereas a server might have just one property which contains its network address.

The actual data associated with a property is stored in the property's value. A user's password, for example, is stored in the property value associated with the password property. An example bindery object and its associated properties and values is shown in the figure below.



Bindery Constraints

The bindery can contain any combination of up to 65,000 bindery objects and properties. Although the bindery's structure is flexible, it should not be overloaded with seldom-used information. A user's mailing list or login preference, for example, might be better stored in the user's mailbox directory. A large bindery is naturally slower to access than a small bindery, and will affect the overall performance of the network. Also, when updates are made to the bindery, there is no attempt to coordinate requests among multiple workstations. Logical locks may be used to coordinate bindery management among workstations.

Bindery Files

Each file server maintains its bindery as hidden files in the SYS:SYSTEM directory. The bindery files are NET$OBJ.SYS, NET$PROP.SYS, and NET$VAL.SYS. The NET$OBJ.SYS file contains objects; NET$PROP.SYS, properties; and NET$VAL.SYS file, property values.

Each file server manages its own bindery files independently of other file servers' binderies. Therefore, the resources and clients available to one file server may not be available to other file servers. Even when similar objects are defined in more than one bindery, each object's unique ID differs from one bindery to another. (An object is given a unique ID when it is created.)

Archiving Bindery Files

It is important to archive the bindery on a regular basis because the bindery files contain information about the file server's clients. However, the bindery files cannot be accessed directly because the file server keeps the bindery files open and locked at all times. To archive the bindery files, the bindery must be closed with the Close Bindery function. The Close Bindery function allows the supervisor, or an object that is security equivalent to the supervisor, to close and unlock the bindery files, thus allowing the bindery to be archived. After the bindery files have been archived, the Open Bindery function is used to give control of the bindery files back to the file server. While the bindery is closed, much of the functionality of the network is disabled. Therefore, the time that the bindery is closed should be kept to a minimum.

Bindery Access

Applications can query the network for information about named network objects through the bindery function calls. In addition to reading the bindery, applications can use bindery function calls to create bindery objects, add properties to bindery objects, and give values to bindery objects' properties. The ability of the application to read from or write to the bindery depends upon the security access levels of both the object and the property to be accessed. The security is enforced by the operating system of the file server where the bindery resides.

Bindery Security

Each file server administers the security for its local resources, services and client accounts through the bindery. The bindery has several levels of security which together provide a flexible yet secure operating environment. Each object and property in the bindery has a security access level associated with it. The security access level controls the read and write access to a bindery object and its properties. Each bindery has a SUPERVISOR object that is granted special bindery security privileges. The supervisor is allowed to grant these special administrative privileges to other objects through the security equivalence feature. The security equivalence feature allows a bindery object to be granted the same access rights as another object. The security equivalence feature is also useful in defining user groups. User groups are a means of logically organizing users into workgroups. This allows the system supervisor to simplify the security process. Generally, a user group is assigned specific directory access rights, then users are added to the user group.

Directory Trustees

The bindery's security and the file system's security are independent. The bindery does not store any of the file system's directory trustee information. Directory trustees are stored in directory entries which are an integral part of NetWare's physical directory structure. The only relationship that the bindery and the file system have is the file system stores each directory's trustee in the form of an object ID. Refer to the File System discussion for more information on file system security and directory trustees.

Security Levels

The bindery's security level controls the read and write access of others to a bindery object or property. Each bindery object has an object security level associated with it, and each property has a property security level associated with it. The object security and the property security are each two nibbles; the low-order nibble controls the read security and the high-order nibble controls the write security. The following chart defines the values for each nibble.

Table 2-1 Security Levels

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the file server.
1	0001	Logged	Access allowed only to clients who have logged in to the file server.
2	0010	Object	Access allowed only to clients who have logged in to the file server with the object's name, type, and password.
3	0011	Supervisor	Access allowed only to clients who have logged in to the server as the supervisor or as an object that has supervisor security equivalence.
4	0100	NetWare	Access only allowed to the NetWare operating system.

For example, a bindery object with an object security level of 31h (supervisor write--logged read) can be viewed by any client that has successfully logged in to the server, but can only have properties added to it by a client that is security equivalent to the supervisor. If this object has a property with a property security level of 33h (supervisor write--supervisor read), the property and its value can only be viewed or modified by a client that is security equivalent to the supervisor.

Supervisor Privileges

Each bindery has an object named SUPERVISOR. The supervisor is the network administrator and is given special access to the bindery. The supervisor, for example, is the only object that can create, delete, or rename bindery objects. It is also the only object that can close and open the bindery for archiving purposes. The supervisor may grant supervisor privileges to other objects through the security equivalence feature.

Security Equivalence

The security equivalence feature allows a bindery object to be granted the same access rights as another object. For example, using security equivalence, the supervisor can grant supervisor rights to other objects. Once one bindery object is created and given detailed security assignments, another object needing the same security can be given security equivalence to the first object.

SECURITY_EQUALS Property

A list of objects that a client is security equivalent to is stored in the client's SECURITY_EQUALS property. This property is used when determining a client's access rights to the file server. When an object logs in to a file server the object's access rights are logically OR'ed together with the access rights possessed by the objects listed in its SECURITY_EQUALS property. When determining an object's directory access rights, the file server's directory parsing algorithm uses only the first 32 objects listed in this property. However, the bindery does allow the SECURITY_EQUALS set to grow beyond 32 objects.

Security equivalences are not transitive. As a result, an object is security equivalent only to the objects explicitly listed in its SECURITY_EQUALS property. In other words, an object's security equivalence is not extended to the equivalences held by an object to which it is security equivalent. For example, if MARY is security equivalent to JOHN and JOHN is security equivalent to SUPERVISOR, MARY is not automatically security equivalent to SUPERVISOR.

User Groups

The security equivalence feature is also useful in defining user groups. User groups are a means of logically organizing users into workgroups. This allows the system supervisor to simplify the security process. Generally, a user group is assigned specific directory access rights, then users are added to the user group.

When a user is added to a user group with the NetWare utility SYSCON, the user's object ID is added to the group's GROUP_MEMBERS property. The group's object ID is then added to the user's GROUP'S_I'M_IN and SECURITY_EQUALS properties.

The group's GROUP_MEMBERS property and the user's GROUPS_I'M_IN property are used together to logically define a group. The SECURITY_EQUALS property is used to ensure that a group member has the directory rights assigned to the group's object.

Bindery Objects

The bindery is a collection of named objects. An object can be a user, user group, file server, print server, print queue, or any other logical or physical entity on the network that has been given a name. Each object has an object name, an object type, an object ID, a dynamic flag, and an object security associated with it. The object name and type uniquely identify the bindery object. The object's ID number, a numeric value that is assigned to the object when it is created, also uniquely identifies the object. The dynamic flag indicates whether the object is dynamic or static. The object security determines whether other objects can access it.

Table 2-2 Object Structure

Field	Type
Object Name	char [0-47]
Object ID	uint32
Object Type	uint16
Properties Flag	uint8
Object State	uint8
Write/Read Security	uint8

Object Name

An object's name is 1 to 47 characters long and must contain only printable characters (21h through 7Dh). Control characters, spaces, slashes (/), backslashes (\), colons (:), semicolons (;), commas (,), asterisks (*), question marks (?), and tildes (~) are invalid characters. Object names are recorded in uppercase in the bindery.

The asterisk (*) and question mark (?) are wild characters and can be used to specify a search pattern when scanning for bindery objects. An asterisk (*) matches 0 or more characters. Thus the pattern "*" will match any object name. A question mark (?) matches exactly one character. Thus the pattern "??" will only match two character object names.

An object is uniquely identified by the object's name and the object's type. Thus, two objects with the same name and different types may exist in the same bindery.

Object ID

Each bindery object is given a unique ID when it is created. This ID is a number which is used as a simple method of identifying an object without specifying the object's name and type. The object ID is only guaranteed to be unique within one file server's bindery. Also, the ID number does not identify the file server's bindery in which an object is defined.

Object Type

The object type identifies the object as a specific type of client or resource. NetWare's currently defined object types are listed in Table 2-3.

Table 2-3 Object Types

Description	Object Type
Unknown	0x0000
User	0x0001
User Group	0x0002
Print Queue	0x0003
File Server	0x0004
Job Server	0x0005
Gateway	0x0006
Print Server	0x0007
Archive Queue	0x0008
Archive Server	0x0009
Job Queue	0x000A
Administration	0x000B
SNA Gateway	0x0021
Remote Bridge Server	0x0024
Synchronization Server	0x002D
Archive Server (Dynamic SAP)	0x002E
Advertizing Print Server	0x0047
Btrieve VAP	0x0050
Print Queue User	0x0053
NVT Server	0x009E
Wild	0xFFFF

The WILD object type (FFFFh) is not an actual object type that would be associated with a bindery object. It is used when scanning the bindery if the actual object type is not known or not relevant to the caller. The WILD object type may not be used when adding an object to the bindery.

Object types up to 8000h are reserved for well-known object types. Other object types may be defined and used by third-party applications as needed. If a general purpose object type is needed, contact Novell's API Consulting Group for assignment of a well-known object type.

Each object type has a set of specifications associated with it. An object created with a defined object type must adhere to the specifications that are defined for that particular object type. For example, an object that is declared as a user must have a password property to login to the file server. A user is also required to have an ACCOUNT_BALANCE property to login if the file server is charging for its services.

Adhering to object type specifications is particularly important for third party value added servers. For example, a server that declares itself as a print server of object type 0007h is expected to provide the same services and have the same client/server protocol as the print server provided by Novell.

Some value added servers need to be recorded in the bindery as both a resource and a client. For example, a server that uses the Service Advertising Protocol (SAP) to advertise its existence on the network, and that also accesses other NetWare API's such as accounting or queue management needs to be defined in the bindery as two separate objects, each with different object types.

One object type would be used for advertising the server, while the other type allows the server to login to the file server as a client and use Accounting or Queue Management. An advertising server is created as a dynamic object, meaning it is an object that is created and deleted frequently. Also, dynamic objects are deleted from the bindery when the file server is initialized. A client object is not dynamic in nature. Clients are static objects that are not created and deleted frequently, but are created once and deleted by the supervisor only when the client no longer needs access to the file server.

Furthermore, the file server does not perform security checks on servers that are advertising. Any server can advertise its services using the Service Advertising Protocol (SAP). The lack of security checks means that an advertising server's access to the file server is very limited. Therefore, the server must also be defined as a client object in each file server's bindery that it needs further access to. In order to gain further access, the value added server must login to the file server as a client.

A number of security checks are performed on the client object before it is allowed access to the file server. Once the client passes the security checks it has access to other services provided by the file server such as file I/O, queue management, and accounting.

A print server is an example of a value added server that requires more than one object type. As indicated in the list of defined NetWare object types, there is an advertising print server (0047h), a print queue (0003h), and a static print server (0007h) object type. All three object types are assigned to and used by a NetWare print server.

The advertising print server is used to advertise that the physical print server is up and running. The print queue is used as a queuing mechanism between the print server and its clients. The static print server type is used by the physical print server to log in to the file server and access its services such as file I/O, queue management, and accounting.

Properties Flag

The properties flag indicates whether one or more properties are associated with an object.

Object State

The object state indicates the expected life time of an object. An object is either static (00h) or dynamic (01h). A static object is a long term object that must be explicitly deleted from the bindery when it is no longer useful. An object of type user is a good example of a static object. A dynamic object is one that is created and deleted frequently, and therefore, is deleted when the file server is brought down and re-initialized. The dynamic flag is used by advertising servers which need to be dynamically added to and deleted from other file servers' binderies.

Object Security

The object security controls the read and write access of others to the bindery object. The low-order nibble determines who can read (scan for and find) the object. The high-order nibble determines who can write to (add properties to or delete properties from) the object. Table 2-4 defines the values for each nibble.

Table 2-4 Object Security Levels

Hex	Binary	Access	Description
0	0000	Anyone	Access allowed to all clients, even if the client has not logged in to the file server.
1	0001	Logged	Access allowed only to clients who have logged in to the file server.
2	0010	Object	Access allowed only to clients who have logged in to the file server with the object's name, type, and password.
3	0011	Supervisor	Access allowed only to clients who have logged in to the server as the supervisor or as an object that has supervisor security equivalence.
4	0100	NetWare	Access only allowed to the NetWare operating system.

For example, an object security of 31h (supervisor write--logged read) indicates that any user logged in to the file server can find the object, but only the supervisor can add a property to the object.

Bindery Properties

Properties are named pieces of information that are attached to objects. Each bindery object may have one or more properties associated with it. For example, the user DAN (object type 0001h) might have associated with it the properties GROUPS_I'M_IN, ACCOUNT_BALANCE, and PASSWORD.

Note that GROUPS_I'M_IN is not the name of a user group to which the object belongs. It is only the name of one category of information associated with that object. In the same way, ACCOUNT_BALANCE is not an actual numerical balance, and PASSWORD is not an actual password. Properties only identify categories of information associated with the object.

Each property has a value associated with it. For example, a value associated with the property GROUPS_I'M_IN must be the object ID of a user group to which DAN belongs. The value of the property ACCOUNT_BALANCE must be user DAN's current balance. The value of the property PASSWORD must be DAN's login password (perhaps COMPILE). Although a property can only have one value, the value can contain multiple segments (each segment being 128 bytes long).

Property Categories

Properties fall into one of two categories: set property or item property. The category determines the type of values the property can have and the number of values.

Set Property

A set property has associated with it a list or set of object IDs that are contained in the property's value. The property value of a set property can consist of multiple segments where each segment may contain up to 32 object IDs. (Each object ID is 4 bytes. Therefore, the maximum number of object IDs that one 128-byte segment can hold is 32.)

A user's GROUPS_I'M_IN property is a set property. The property value associated with the GROUPS_I'M_IN property contains the object IDs of the groups to which the user, DAN (in the above example), belongs. The values of set properties are always object IDs grouped into one or more 128-byte segments. It is important that set property values do not contain anything other than object IDs because the operating system interprets each segment of a set property value to be an array of object IDs.

When an object is deleted from a set property, the operating system searches consecutive segments of the property's value for an object ID that matches the object ID of the member to be deleted. When the member is found it is deleted. The remaining IDs in the segment are shifted and the last previously used slot in the segment is filled with zeros. This ensures that IDs within a segment are packed. However, IDs are not packed between segments.

Item Property

An item property has associated with it a property value which can contain any type of data (typically contains a numeric value, a string or a structure). The bindery attaches no significance to the contents of a item property's value. An item property's value is defined and interpreted by APIs and by application programs not by the bindery process.

A user's PASSWORD and ACCOUNT_BALANCE properties are both examples of item properties. The PASSWORD property is defined to have only one segment and contains an encrypted password. The ACCOUNT_BALANCE property value contains a monetary balance in the first 4 bytes and a credit limit in the next 4 bytes of the 128-byte segment. The rest of the segment is filled with zeros.

Property Fields

Each property has a property name, property state, property type, and a property security associated with it. The property name identifies an object's property. The property flags field contains two flags: the static/dynamic flag and the item/set flag. The static/dynamic flag indicates the expected life time of a property. The item/set flag specifies the type of information that is stored in the property's value. The property security determines who has access to the property.

Property Name

A property's name is 1 to 15 characters long and must contain only printable characters (21h through 7Dh). Control characters, spaces, slashes (/), backslashes (\), colons (:), semicolons (;), commas (,), asterisks (*), question marks (?), and tildes (~) are invalid characters. Property names are recorded in uppercase in the bindery.

The asterisk (*) and question mark (?) are wild characters and can be used to specify a search pattern when scanning a bindery object's properties. An asterisk (*) matches 0 or more characters. Thus the pattern "*" will match any property name. A question mark (?) matches exactly one character. Thus the pattern "??" will only match two character property names.

Property State

The property state flag indicates the expected life time of a property. A property is either static or dynamic. A static property is a long term property that must be explicitly deleted from the bindery when it is no longer useful. The ACCOUNT_BALANCE property is an example of a static property. A dynamic property is one that is created and deleted frequently, and therefore, is deleted when the file server is brought down and re-initialized. The ACCOUNT_HOLDS property is an example of a dynamic property.

Bit 0 is the static/dynamic flag and is defined as follows:

7	6	5	4	3	2	1	0	
							0	Static property
							1	Dynamic property

Property Type

The item/set flag indicates whether the property's value contains an item or a set of object IDs. The contents of item property values are defined and interpreted by applications or by application program interfaces (APIs). The contents of set property values are interpreted by the bindery process as a series of object ID numbers, each 4 bytes long.

Bit 1 is the item/set flag and is defined as follows:

7	6	5	4	3	2	1	0	
						0		Item property
						1		Set property

Property Security

The property security controls the read and write access of others to the property. The low-order nibble determines who can read (scan for and find) the property. The high-order nibble determines who can write to (modify) the property's value. See Table 2-4, above, for definitions of the values for each nibble.

NetWare Properties

NetWare has defined many different properties for many different purposes. Some properties contain further information about an object while others are used to administer the network security system. All properties defined by Novell which contain numeric data (including set properties) are stored in hi-lo order.

Table 2-5 lists all of the properties defined by NetWare. The bindery property descriptions follow the table. The other properties are described in the appropriate chapter for the API (for example, the Accounting properties are described in Chapter 1, "Accounting Services").

Table 2-5 Properties Defined by NetWare

Property Name	Object Type	Flags	Security Write	Security Read	Application Program Interface (API)
ACCOUNT_BALANCE	User	Static Item	3	2	Accounting
ACCOUNT_HOLDS	User	Dynamic Item	3	2	Accounting
ACCOUNT_SERVERS	File Server	Static Set	3	1	Accounting
ACCT_LOCKOUT	File Server	Static Item	3	3	Security
BLOCKS_READ	File Server	Static Item	3	1	Accounting
BLOCKS_WRITTEN	File Server	Static Item	3	1	Accounting
CONNECT_TIME	File Server	Static Item	3	1	Accounting
DISK_STORAGE	File Server	Static Item	3	1	Accounting
GROUP_MEMBERS	User Group	Static Set	3	1	Bindery
GROUPS_I'M_IN	User	Static Set	3	1	Bindery
IDENTIFICATION	User	Static Item	3	1	Bindery
LOGIN_CONTROL	User	Static Item	3	2	Security
NET_ADDRESS	File Server	Dynamic Item	4	0	Service Advertising
NODE_CONTROL	User	Static Item	3	2	Security
OLD_PASSWORDS	User	Static Item	3	3	Security
OPERATORS	File Server	Static Set	3	3	Bindery
PASSWORD	User	Static Item	4	4	Bindery
Q_DIRECTORY	Print Queue	Static Item	3	3	Queue Management
Q_OPERATORS	Print Queue	Static Set	3	1	Queue Management
Q_SERVERS	Print Queue	Static Set	3	1	Queue Management
Q_USERS	Print Queue	Static Set	3	1	Queue Management
REQUESTS_MADE	File Server	Static Item	3	1	Accounting
SECURITY_EQUALS	User	Static Set	3	2	Bindery
USER_DEFAULTS	Supervisor	Static Item	3	1	Security

GROUP_MEMBERS Property. The GROUP_MEMBERS property contains a list of users that are members of a user group. This property is attached to user group objects. The GROUP_MEMBERS property is a static/set property and has object read and supervisor write security access levels.

GROUPS_I'M_IN Property. The GROUPS_I'M_IN property contains a list of user groups that a user is a member of. This property is attached to user objects. The GROUPS_I'M_IN property is a static/set property and has logged read and supervisor write security access levels.

IDENTIFICATION Property. The IDENTIFICATION property contains a user or user group's full name. This property is attached to user and user group objects. The IDENTIFICATION property is a static/item property and has logged read and supervisor write security access levels.

OPERATORS Property. The OPERATORS property contains a list of objects that are authorized file server console operators. This property is attached to the file server's object. The OPERATORS property is a static/set property and has supervisor read and supervisor write security access levels.

PASSWORD Property. The PASSWORD property contains an object's encrypted password. This property is attached to user objects. Any object that logs in to the file server is required to have the PASSWORD property. The PASSWORD property can be created with the Change Bindery Object Password function call. The PASSWORD property is a static/item property and has NetWare read and NetWare write security access levels.

SECURITY_EQUALS Property. The SECURITY_EQUALS property contains a list of objects that an object is security equivalent to. This property is attached to user and user group objects. The SECURITY_EQUALS property is a static/set property and has object read and supervisor write security access levels.

End of Chapter

Chapter 3
Connection Services

This chapter covers Connection services. Connection Services are used to create connections to file servers and return information about those connections and the clients that made them.

Note: These procedures are not necessary for previous versions of the DOS client APIs; connections are maintained by the DOS shell with which the APIs communicate. They are, however, necessary for NetWare for AViiON Systems.

This chapter is divided into the following sections:

- Attaching and detaching
- Logging in and logging out
- Connection Tables

Attaching and Detaching

In order to access and manage NetWare file servers and their associated network resources, an API client must first open a transport with, and then attach to, a file server. The call, NWAttachToServerPlatform, performs both of these functions; however, the client can use the call NWOpenTransport to open the transport separately, before using NWAttachToServerPlatform. NWAttachToServerPlatform searches the network for the nearest server. Once the API call finds a server, the API call searches the bindery for the specified file server.

If the API call finds the address and name of the specified server in the bindery, the API call places the address and name of the server in a Client Connection Table. The API call then sends a packet to the specified file server. If the file server responds with an available connection slot, the attachment is successful. The call, NWAttachToServerPlatform, returns a file server connection number (serverConnID) to the client.

Once an API client has established a file server attachment, that attachment remains valid until the API client breaks that attachment or the connection is broken for the client. When the connection is broken, the client's object ID and client connection number (clientConnID) are removed from the file server's Connection Information Table. That connection slot then becomes available to other clients who want to attach.

The calls, NWDetachFromServerPlatform and NWClearConnID can be used to remove a clientConnID from the file server's Connection Information Table, and thus break the connection.

The NetWare watchdog will also clear a client's connection when the client does not respond to the watchdog packets. If the watchdog has not heard from a client in five minutes, the watchdog sends a watchdog packet to the client. If the client does not respond, the watchdog starts sending watchdog packets at one-minute intervals for up to ten minutes. If the client does not respond by the end of the fifteen minutes, the watchdog assumes that the client is no longer connected to the file server and clears the client's connection.

Once a client has been detached, the client must then re-attach before the client can log in again.

Logging in and logging out

After attaching to a file server, an application must log in to the file server by using NWLoginToServerPlatform. Logging in allows the client to gain the rights and privileges necessary to manage and manipulate network resources. To login to a file server, an application must provide a bindery object's name, type, and password. When an object logs in to a file server, the file server puts the object's ID number in the file server's Password table and in the Connection Information Table.

When an object logs out of a file server, the file server removes the object's ID from the file server's Password table. However, the object's ID is not removed from the Connection Information Table until the object detaches from the file server.

Connection Tables

Two connection tables are kept, one by the APIs and the other by the file server.

Client Connection Table

The Client Connection table is kept by the APIs for each client attached to the file server. Various APIs allow the client to query the table for the information listed below. The other fields in the table are for internal use only and contain information such as packet size and the number of packet retries.

This table contains the following information that APIs can query.

File server connection number (serverConnID). This field contains the file server connection number. The APIs assigns this number (0 through 7) as the client attaches to file servers. The first file server, or the client's default file server, is assigned 0; the next file server, 1; and so forth. If the client has already attached to 8 file servers, the attempt to attach to a ninth file server fails.

File server name. This field contains the name of the file server that the client is attached or logged into.

NetWare version. This field contains the version of the operating system that the file server is running.

Client connection number (clientConnID). This field contains the client connection number. The file server assigns this number (0 through 250) as the file server grants a connection slot to a client. If all of the file servers connection slots have been assigned to other clients, the attempt to attach fails. The number of slots varies with the version of the operating system. NetWare for AViiON Systems supports up to 250 connection slots, depending upon your license agreement.

Connection Information Table

The file server maintains the Connection Information table. This table contains all clients attached or logged into the file server. The APIs use the client's clientConnID and serverConnID to query the table for the following information.

Object Name (clientObjectName). This field contains the object's name. Each object must be created on the file server before the object can log in. (For example, you can create users with SYSCON or print queues with PCONSOLE.) Each object must be given a unique name for its type. This name becomes the clientObjectName that must be passed by the API for a client to log in.

Object ID (clientObjectID). This field contains the object ID. As each object is created, the file server assigns the object a unique object ID. This number is used to identify the object without specifying the object's name and type.

Object Type (clientObjectType). This field contains the object type. As each object is created, the object must be assigned a bindery type. When a user is created with SYSCON, SYSCON assigns type User to the object. When a group is created with SYSCON, SYSCON assigns type Group to the object. See Chapter 2, "Bindery Services," for a list of possible bindery types.

Login Time (clientLoginTime). This field contains the date and time that the object logged in to the file server.

Network address and Node address (internetAddress). This field contains the unique address of the client. This address is made up of the client's network address and node address. The network address is the address that has been assigned to the cabling system the client is physically cabled to. The node address is the address of the network board installed in the client's workstation.

Using Client Connection IDs

The clientConnID (or client connection number) indicates the number allocated by the file server to a particular client. It is important not to confuse clientConnID with serverConnID. These numbers can provide an easy way to identify objects logged in on the network and obtain additional information about them. The API call, NWGetConnectionInformation, uses both (serverConnId and clientConnID) to return the object name, type, ID and login time about the object that has made the connection. Likewise, the API call, NWGetInternetAddress, uses both (serverConnID and clientConnID) to return the network and node address a connection is using.

A list of all the connections for a particular bindery object can be obtained with NWGetObjectClientConnIDs. To make this API call, an application must pass in the object's name and type. A client can obtain its own connection number with the default file server with NWGetClientConnID.

End of Chapter

Chapter 4
File Services

File Services enable applications to manipulate file system information. Because NetWare for AViiON Systems function calls work independent of the NetWare client shell, the file system service APIs provide many of the functions that would normally be taken care of by the NetWare client shell. This chapter is divided into the following sections:

- File Identification
- Security
- File Manipulation
- Directory Manipulation
- Volume Manipulation
- Trustees
- Salvageable Files

File Identification

File Service APIs identify files by using directory handles, file handles, paths, wildcards, entry IDs, and search attributes. Each is described below.

Directory Handles

A directory handle is a number from 0 to 255 that points to a volume or directory and each workstation maintains a table of directory handles. Directory handles are created with the Path Service APIs. (For more information on creating directory handles, see Chapter 5, Path Services.)

Directory handles can be used in the following ways to identify a file:

- When a specific directory handle is known, a file can be identified by passing the directory handle and the file name.
- When a zero is passed as the directory handle, a file can be identified by passing the full path as the file's name.

See "Paths" on the following page for proper syntax.

File Handles

NetWare creates file handles when a file is opened or created. The file handle is a pointer to the location of the file. The APIs that either create or open the file are responsible for creating the handle and passing it to the application. When the application closes the file, the application should return the file handle so that the API can deleted it from the workstation's table.

Paths

When specifying a file to a file services call, an application can use either a full or a partial file path. A full file path has the following format:

Volume:\Directory\Directory\File

Because a full file path completely specifies a file, file services calls ignore the value of their directory handle parameter when an application passes a full file path. When a full path is specified, set the directory handle to zero.

A partial file path includes a file name and optionally one or more antecedent directory names. The file service routine then combines the directory handle setting and the partial file path to obtain a full file specification. For example, suppose an application calls a file services routine and passes a directory handle mapped to the directory WORK:\HOME\MARY and a file path parameter containing the partial file path ACCTS\ACTIVE\ZZYZX.DAT. The routine would use the directory handle and partial file path to identify the file:

WORK:\HOME\MARY\ ACCTS\ACTIVE\ZZYZX.DAT.

Wildcards

Some file services calls accept wildcard characters in file names. To see which calls provide wild card functionality, see *NetWare® for AViiON® Systems: C Interface Reference Guide*. NetWare supports the following wildcard characters:

Character	Name
*	Asterisk
?	Question Mark
^*	Augmented Asterisk
^?	Augmented Question Mark
^.	Augmented Period

Each wildcard character is described below.

Asterisk. An asterisk matches zero or more characters. The pattern * therefore matches any string. The pattern *.* matches anything with a period. (It does not match names without a period, since a period is treated as any other character).

Question Mark. A question mark matches exactly one character, even a period.

Augmented Asterisk. The augmented asterisk is an asterisk with its high-order bit set. It matches any character except a period.

Augmented Question Mark. An augmented question mark is a question mark with its high-order bitset. It matches one character or an end-of-string.

Augmented Period. The augmented period is a period with its high-order bit set. This character matches a period or an end-of-string. The augmented period is included primarily to allow a shell or program to search for file names while ignoring trailing periods.

Table 4-1 gives examples of wildcard strings and file names that would and would not match them.

Table 4-1 Wildcard Patterns

Character	Example	Matches	Does Not Match
*	*	All files	
	.	A.B	AB
?	A?C	ABC	AC
		A.C	ABBC
^*	^*	ABC	A.B
		FILE	
	A^*C	AC	ABC.C
		ABBC	
^?	A^?C	ABC	A.C
		AC	
	A^?^?	A	A.B
		AB	
		ABC	
^.	AB^.	AB	ABC
		AB.	AB.C

Entry IDs

Entry IDs are used with many of the file system API calls. It is used to signal the file server to search for the file. The application should never have to use or manipulate this value, except for the first time the call is used.

When a call uses an entry ID (the entryID parameter), the parameter should be initialized to -1. For subsequent calls, the entryID value will be filled in by the file server and should not be altered. This value is an internal number used by the file server, which represents the previously-scanned entry.

Search Attributes

Some File Service functions (NWDeleteFile, NWScanFileEntryInfo, and NWSetFileEntryInfo) act on normal, hidden, or system files, depending upon the setting of a search attributes parameter. These functions always act on normal files, whether the hidden or system bit is set or not. The search attribute parameter does NOT set a file or directory's attributes, it merely indicates to the file server the type of file or directory being looked at.

The search attributes for a given function call are compared with the file or directory's attribute byte. For example, calling NWScanFileEntryInfo with a search attribute of system will allow files to be scanned that are either normal or that have the system bit (in their attribute byte) turned on.

To assign various attributes to files or directories requires the use of NWSetFileAttributes, NWSetDirEntryInfo, or NWSetFileEntryInfo. When using search attributes in conjunction with setting attributes, the search attributes should reflect the file or directory as it currently exists, not as the file or directory will be after the attributes are set.

Security

All file service APIs require security checking before the function is completed. If the requesting client fails the security check, the function also fails. The functions check for one or more of the following: bindery security, effective rights, and attributes. The following examples illustrate security checking.

Bindery security	Some APIs require the client to have read and write privileges to bindery objects. For example, the API that sets volume restrictions (NWSetObjectVolRestriction), checks to make sure the client has a bindery access level of supervisor equivalence for the object. If the client does not have supervisor equivalence, the call fails.
Effective rights	Some APIs require the client to have specific rights. For example, the API that deletes a file (NWDeleteFile) checks to make sure that the client has Erase rights to the file. If the client does not have Erase rights to the file, the call to delete the file fails.
Attributes	Some APIs require the file to have specific attributes turned off. The API that deletes a file checks the file's current attributes. If the Delete Inhibit attribute has been set, the file cannot be deleted. Therefore the call to delete the file fails.

Each of type of security check is described in greater detail on the following pages.

Bindery Security

Bindery security controls the read and write access to bindery objects and bindery properties. Each bindery object has an object security level associated with it, and each property has a property security level associated with it.

Both objects and properties use the security access levels that are shown in Table 4-2.

Table 4-2 Security Levels

Access	Description
Anyone	Access allowed to all clients, even if the client has not logged in to the file server. Both a read and a write bit can be set.
Logged	Access allowed only to clients who have logged in to the file server. Both a read and a write bit can be set.
Object	Access allowed only to clients who have logged in to the file server with the object's name, type and password. Both a read and a write bit can be set.
Supervisor	Access allowed only to clients who have logged in to the server as the supervisor or as an object that has supervisor security equivalence. Both a read and a write bit can be set.
NetWare	Access only allowed to the NetWare operating system.

For more information, see Chapter 2, Bindery Services.

Effective Rights

Effective rights are the rights a user can actually exercise in a given directory or file. Effective rights for NetWare 2.x differ from those for NetWare 3.x and NetWare for AViiON Systems. For NetWare 2.x, see Appendix A, *NetWare® for AViiON® Systems: C Interface Reference Guide*.

To determine a user's effective rights in NetWare 3.x and NetWare for AViiON Systems, you must know what rights were granted in trustee assignments and what the directory or file's Inherited Rights Mask is.

The following three sections define the effects of each right, the effects of trustee assignments, and the effects of the Inherited Rights Mask.

Rights

Both trustee assignments and Inherited Rights Masks use the same eight trustee rights to control access to directories and files. The eight rights are shown in the list below. Each right is represented by its initial.

S Supervisory	C Create	F File Scan
R Read	E Erase	A Access Control
W Write	M Modify	

Each of these is defined below.

Supervisory. Grants all rights to the directory, its files and subdirectories. The Supervisory right overrides any restrictions placed on subdirectories or files with an Inherited Rights Mask. Clients who have this rights in a directory can grant other clients Supervisory rights to the directory, its files and its subdirectories. Once the Supervisory rights has been granted, it can be revoked only from the directory to which it was granted. It cannot be removed from the Inherited Rights Mask.

Read. Grants the right to open and read the file.

Write. Grants the right to open and write to the file. Clients usually need the Modify right also because writing to a file requires modification to the file's attributes.

Create. Grants the right to create files and directories and to salvage files.

Erase. Grants the right to delete directories and files. All delete functions require the user to have this right. One create function (NWCreateFile) requires the client to have this right also because the function deletes existing files with the same name as the new file.

Modify. Grants the right to change directory and file attributes. It also grants the right to rename directories and files. It does not grant the right to modify the contents of a file. The right to modify the contents of a file is granted with the Write right.

File Scan. Grants the right to see directories and files. It also grants the right to see from the point the right is assigned up the directory structure to the root.

Access Control. Grants the right to modify a directory's or a file's trustee assignments and Inherited Rights Mask.

Trustee Assignments

Trustee assignments are rights granted to specific users (or groups) that allow trustees to use a file or a directory in particular ways, for example, only for reading. The network manager can select the appropriate rights to assign to users or groups in each directory or file.

A trustee assignment automatically grants users the right to see to the root of a directory. However, users can't see other subdirectories in the directory unless they have been granted rights in those subdirectories or have been granted the Supervisory right in the directory.

Trustee assignments allow rights to flow down the directory structure. In other words, users will usually have the same rights in a subdirectory or file as they were granted in the parent directory. However, if the sudirectory's or file's Inherited Rights Mask is modified, their effective rights can change.

Inherited Rights Mask

An Inherited Rights Mask is given to each file or directory when created. Although the default Inherited Rights Mask includes all rights, users can only exercise the rights they have been granted in trustee assignments.

An Inherited Rights Mask only affects users' rights in a level below the point they were granted trustee assignments. If the Inherited Rights Mask is modified for a file or a subdirectory below the original trustee assignment, the only rights the user can inherit for the file or the subdirectory are rights that are allowed by the Inherited Rights Mask. The rights in the Inherited Rights Mask are AND'ed to the effective rights of the parent directory.

For example, if Ann is granted the Read right with a directory trustee assignment but the Read right is revoked by the Inherited Rights Mask at the subdirectory level, Ann cannot read files in the subdirectory.

Attributes

Attributes assign special properties to files and directories. Attributes override rights and prevent tasks that effective rights would allow. They can be used to restrict or inhibit copying, deleting, renaming, writing, and sharing. The list below shows the specific directory and file attributes and their associated letter representation.

Directory Attributes	Letter Representation
Delete Inhibit	D
Hidden	H
Purge	P
Rename Inhibit	R
System	Sy

File Attributes	Letter Representation
Archive Needed	A
Copy Inhibit	C
Delete Inhibit	D
Execute Only	X
Hidden	H
Indexed	I
Purge	P
Read Audit	Ra
Read Only / Read Write	Ro / RW
Rename Inhibit	R
Shareable	S
System	Sy
Transactional	T
Write Audit	Wa

File Manipulation

The File Service APIs allow clients to manipulate files. They can do the following: create files, delete files, open files, close files, read from files, write to files, move files, get and set file attributes, scan and set file information, set file inherited rights mask, scan and set file trustee assignments. Each task is described below except scan and set trustee assignments, which are discussed later in this chapter.

Copying and Moving Files

NetWare has three APIs that move or copy files: NWFileCopy, NWMoveFile, and NWMoveEntry.

The NWFileCopy function copies network files. Both the source and destination files must reside on the same file server.

The NWMoveFile function moves or renames a file. Both the source and destination files must reside on the same file server and the same volume.

The NWMoveEntry function moves or renames either files or directories.

To copy a file from one file server to another, the application must create the destination file (NWCreateFile or NWCreateNewFile), read from the source file (NWReadFile), and then write to the destination file (NWWriteFile).

Creating Files

Two function calls create files: NWCreateFile and NWCreateNewFile. The NWCreateFile function will create a new file name and also allows for overwriting an existing file of the same name. The NWCreateNewFile function only creates a new file name, and this request will always fail if a file with the same name already exists. Users need Create rights to use NWCreateNewFile and Create and Delete rights to use NWCreateFile.

Creating a file doesn't allocate any space. The file size will be zero until data is actually written to the file. The minimum file size is 4KB.

Deleting Files

The NWDeleteFile function marks specified files for deletion. Files are specified by passing a directory handle and file path (which can include wildcards) to the function. The NWPurgeSalvageableFile function actually deletes all files that a workstation has marked for deletion. Until files that have been marked for deletion are purged, they can be restored by calling the NWRecoverSalvageableFile function. The NWDeleteFile function corresponds to the DOS ERASE or DEL commands.

Opening and Closing Files

The NWOpenFile function allows a client to open an existing file for reading or writing. This function must be called before calling the read or write function. The client must have Read rights to open and read an existing file or Write rights to open and write to an exiting file.

The NWCloseFile function closes a file after it has been opened with NWOpenFile.

Reading from and Writing to Files

The NWReadFile function allows you to read a file. The NWWriteFile function allows you to write to a file.

The NWReadFile and NWWriteFile calls require a file handle to the file which will be read from or written to. This file handle is obtained by either opening or creating a file. This file handle points to a file located on a NetWare file server.

The client must have Read rights to read a file and Write rights to write to a file. The client usually needs to have Modify rights to write to a file because most write operations change the file's attributes.

Getting and Setting File Attributes

NetWare has four functions that allow manipulation of a file attributes. Two functions, NWScanFileEntryInfo and NWSetFileEntryInfo, are discussed in the next section. These two allow you to scan and set file attributes as well as many other file information fields.

The NWGetFileAttributes function returns a specified file's attributes. The NWSetFileAttributes allows a client to modify a file's attributes to a specified attribute value. The client must have Modify rights in to the specified file, and the file must not be in use by other clients. This call supports wildcard attribute setting.

One attribute, the Execute Only attribute, cannot be reset. Since the attribute cannot be reset, setting this attribute requires the client to have supervisor equivalence.

See the section, "Attributes" later in this chapter for a list of supported attributes.

Scanning and Setting File Information

NetWare has two functions that allow manipulation of file information: NWScanFileEntryInfo and NWSetFileEntryInfo.

The NWScanFileEntryInfo function returns information about the file, and the NWSetFileEntryInfo function sets information kept on files. This information can be seen in the NWFileEntryInfo_t structure. A pointer to the entire structure should be passed, even if only one item is being scanned or changed.

Some items have a separate function that allows you to scan or change only that item. These items are shown in the table below.

Table 4-3 Security Levels

Item	Scan Function	Set Function
Attributes	NWGetFileAttributes	NWSetFileAttributes
Inherited Rights Mask		NWSetFilesInheritedRightsMask
Name		NWMoveFile
Trustees	NWGetEntrysTrustees	NWSetTrustee

Since the functions listed above are designed to scan or change one item, we recommend using them if you have only the specific item to change or scan. Use the NWScanFileEntryInfo and NWSetFileEntryInfo when you need to change multiple items.

Using NWSetFileEntryInfo requires more than passing a pointer to the entire file structure. You must also pass change attributes which correspond to the data being changed. For example, if the owner of the file is being changed, then the entire structure would be allocated and the new ownerID would be put in the ownerID field of the structure. Correspondingly, the NWCA_OWNER_ID change attribute would be passed in the changeAttributes parameter.

The client's security requirements change with the item being changed. The date and time fields and ID fields require the client to have supervisor equivalence. (These fields are described in the "IDs and Users" and "Date and Time Information" sections.) Other items require the rights shown in the table below.

Table 4-4 Client Security Requirements

Item	Rights
Attributes	Access Control except for Execute Only. Execute Only requires the client to have supervisor equivalence.
Inherited Rights Mask	Access Control
Name	Modify

IDs and Users

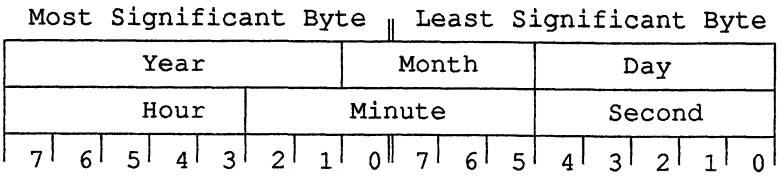
Some structure members of the file system API calls are bindery object IDs of users. These object IDs are used to identify the following: archiverID, ownerID, updatorID, trusteeID, and objectID. If you would like to know the object name associated with those IDs, the object name can be obtained by calling the NWGetObjectName function.

The operating system automatically fills these fields with the object ID of the user who performs the task. For example, the ownerID is the user who created the file. A client must have supervisor equivalence to set the owner of the file to another user.

Date and Time Information

Several of the file system API calls, such as NWGetDirEntryInfo and NWScanFileEntryInfo receive time and date information for creation, last modification, last archive, and last access. When a user performs these operations, the operating system automatically fills the fields with the system the date and time. A client must have supervisor equivalence to change the date and time.

The date and time are stored in a four byte value (uint32); the format is shown below.



The first two bytes of the four byte value contain the year, month, and day. The last two bytes contain the hour, minute, and second.

Year. The year field is the seven high-order bits of the most significant byte. To obtain the actual year, add 1980 to the value in the year field.

Month. The month field comprises the low-order bit of the most significant byte and the three high order bits of the least significant byte. This field is a number from 1 to 12.

Day. The day field is the four low-order bits of the least significant byte. This field is a number from 1 to 31.

Hour. The hour field is the five high-order bits of the third byte.

Minute. The minute field is the three low-order bits of the third byte and the three high-order bits of the fourth byte.

Seconds. The seconds field is the five low-order bits of the fourth byte and contains a value from 0 to 29; multiply by 2 to obtain the actual second's value.

Setting the File's Inherited Rights Mask

The inherited rights masks are given to each file and directory at creation. The file inherited rights mask controls which directory rights can be exercised on the file. See "Effective Rights," in this chapter, for additional information on rights.

The APIs allow you to set the rights mask for a file. The rights you pass overlay the existing inherited rights mask. You cannot add a right by passing in only one right; you must pass in all rights that you want set in the inherited rights mask.

The client must have Access Control rights to the file to set the inherited rights mask.

Getting Information on File Name Spaces

The NWGetNameSpaceInfo function returns all name spaces and data stream information for the specified file server and volume. This call is valid for NetWare for AViiON Systems and NetWare version 3.x. This function passes three parameters.

```
NWGetNameSpaceInfo( serverConnID, volNum, &nameSpace )
```

Directory Manipulation

The File Service APIs allow clients to manipulate directories. They can do the following: create directories, delete directories, rename directories, scan and set directory information, set directory inherited rights mask, get and set directory restrictions, and scan and set directory trustees. Each task is described below except scan and set directory trustees. See the section, "Trustees," later in this chapter for additional information.

Creating Directories

The NWCreateDir function creates a directory on the server specified by the connection ID. This function passes a pointer to the structure containing the directory handle, file server connection ID, and a pointer to the path name. It also passes the Inherited Rights Mask for the new directory. The dirHandle parameter can be zero if the dirPath parameter contains the complete path of the new directory, including the volume name. This call will not accept wild card characters. Also the requesting client must have Create and Parental rights in the directory that will become the parent directory.

This call will not sequentially create a string of directories; this call only creates the last directory provided in the NWPath_t structure provided by the client. It creates only one directory at a time.

This call differs from NWCreateFile in that a handle is not returned. To obtain a directory handle to this directory, you must use NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

Deleting Directories

The NWDeleteDir function deletes a NetWare directory. The call expects a pointer to the NWPath_t structure containing the directory handle, file server connection ID, and the path name. This function will fail if the directory does not exist or there are files in the existing directory. This function automatically deallocates any directory handles pointing to the directory, but will fail if another client has the directory handle pointing to the directory.

Renaming Directories

The NWRenameDir function allows you to change the name of a directory. This call does not move a directory to a new location. NWRenameDir expects a pointer to the NWPAth_t structure containing the directory handle, file server connection ID, and the path name. It also passes a pointer to the array allocated for the new directory name.

The newDirName parameter should only contain the directory's new name, not a path specification. Names longer than the DOS 8.3 will be truncated.

If the application is moving a directory, NWMoveEntry should be used.

Scanning Directory Information

The NWScanDirEntryInfo function scans for directory entry information such as entry name, attributes, and creation, archive, last modification date and time.

The function passes four parameters as shown below.

```
NWScanDirEntryInfo( &path, &entryID, searchAttributes, &dirInfo )
```

The application must provide a search string in the pathName field of the NWPath_t structure such as SYS:APPS*. If all directories are scanned, the pathName field in the NWPath_t structure should contain ...*. If, for example, only directories beginning with "t" are being scanned, pathName should contain ...\t*.

Getting Directory Information

Two function calls allow you to get directory information. The NWGetDirEntryInfo function provides information about a directory through the directory handle. The correct syntax is shown below.

```
NWGetDirEntryInfo( serverConnID, dirHandle, &dirInfo )
```

This call is valid for NetWare for AViiON Systems and NetWare version 3.x. It is useful for obtaining information from the root directory. The dirHandle parameter must be allocated using the NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle call.

Another function, NWGetDirRestriction, obtains restriction information, such as maximum blocks and available blocks. This information is obtained for the specified directory and the directories above it. The restrictions are set using NWSetDirRestriction. Five parameters are being passed in this function.

```
NWGetDirRestriction( serverConnID, dirHandle, &numberOfEntries,
    &restrictions, maxListElements )
```

This function scans for the amount of disk space assigned to all directories between the current directory (referenced by the dirHandle) and the root directory. To find the actual amount of space available to a directory, scan all the entries returned in the restriction array and use the smallest one. Directories which have no restrictions will not return any information. If no entries are returned, no space restrictions exist for the specified directory.

To calculate the amount of space in use, simply subtract availableBlocks from maxBlocks. If maxBlocks is a negative value, the limit is 0. If the availableBlocks value is negative, the available blocks is 0.

This function is available for NetWare for AViiON Systems and NetWare version 3.x. You must allocate a dirHandle before you make this call.

Setting Directory Information

Two functions allow for setting directory information. The NWSetDirEntryInfo function sets information kept on directories. This information can be seen in the NWDirEntryInfo_t structure. A pointer to the entire structure should be passed, even if only one item is being changed. Furthermore, the change attributes must be passed which correspond to the data being changed; these attributes can be OR'ed together to make several changes. For example, if the owner of the directory

is being changed, the entire structure would then be allocated and the new ownerID would be put in the ownerID field of the structure. Correspondingly, the NWCA_OWNER_ID change attribute would be passed in the changeAttributes parameter.

The NWSetDirRestriction function sets or clears a directory's restrictions. This function requires that the application pass an allocated directory handle to the directory to which the restrictions apply.

The restriction parameter passes a 0 to clear all restrictions or a number corresponding to the space available in the directory. Restrictions are in 4K blocks, therefore, a restriction of 1 will restrict the space usage in a particular directory to 4K.

This call is valid for NetWare for AViiON Systems and NetWare version 3.x. The directory handle can be obtained by calling either NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

Setting the Directory's Inherited Rights Mask

The NWSetDirsInheritedRightsMask function modifies the rights mask for a directory path. It passes three parameters as shown below.

```
NWSetDirsInheritedRightsMask( &path, revokeRightsMask,
    grantRightsMask )
```

This function will revoke directory rights by removing from the inherited rights mask any rights the application passes in with the revokeRightsMask parameter. It will grant rights by adding to the file's inherited rights mask any rights passed in with the grantRightsMask parameter.

Volumes

A NetWare 3.x file server can have up to 32 disk volumes installed. NetWare for AViiON Systems has a configurable number of volumes. NetWare provides eight calls for obtaining information about volumes.

Clearing Volume Restrictions

The NWClearObjectVolRestriction function clears any volume restrictions placed on an object by the NWSetObjectVolRestriction call. This function passes three parameters.

```
NWClearObjectVolRestriction( serverConnID, volNum, objectID )
```

This function is only valid on NetWare 3.x.

Getting Volume Restrictions

Two file system API calls get the restriction information associated with a volume. These two functions are only valid on NetWare 3.x.

The NWGetObjectVolRestriction function acquires the volume restrictions that are placed on a specified object (such as a user). The function passes five parameters.

```
NWGetObjectVolRestriction( serverConnID, volNum, objectID, &restriction,
    &inUse )
```

This function returns the amount of space restriction based on 4K blocks on a specified object as well as the current amount of space used by the object. There are no restrictions if the restriction value returned is 0x40000000.

The NWGetVolsObjectRestrictions call will get any object restrictions on a volume. It passes five parameters (see the example below).

```
NWGetVolsObjectRestrictions( serverConnID, volNum, numberOfEntries,
    &restrictions, &maxListElements )
```

This call returns the number of entries that were copied into the restrictions array (0 - n) and the user restrictions.

Setting Volume Restrictions

The NWSetObjectVolRestriction function sets the volume restrictions on a specified object (such as a user). The function passes four parameters.

```
NWSetObjectVolRestriction( serverConnID, volNum, objectID, restriction )
```

This function is similar to NWSetDirRestriction in that a space restriction is set, but the restriction applies to a specific object rather than overall. Restrictions are set in 4Kbyte blocks. This call is only valid with NetWare versions 3.x. The volume number can be obtained by calling NWGetVolNum.

Getting Volume Numbers

The NWGetVolNum function returns the volume number based on the file server connection ID number and the volume name. The volume name cannot contain wildcards. If the volNum parameter is between 0 and the maximum allowable volume number on the network, the call is successful and a zero is returned. This call passes three parameters as shown below.

```
NWGetVolNum( serverConnID, volName, &volNum )
```

The NWGetVolNum is the inverse of NWGetVolName, returning a volume's slot number, given its name.

Getting Volume Names

The NWGetVolName function returns the name of a volume (a string of up to 16 characters) given a specified volume number [0..31]. The volNum parameter identifies the volume on the file server's Volume table. The Volume table contains information about each volume on the file server. The volName parameter is 16 bytes long (including a null-byte). A volume name can be from 1 to 16 characters long and cannot include spaces or special characters such as *, ?, :, /, and \. This call passes three parameters as shown below.

```
NWGetVolName( serverConnID, volNum, volName )
```

Getting Volume Information

Two calls get information about a volume: NWGetVolInfoWithHandle and NWGetVolUsage. The NWGetVolInfoWithHandle function returns directory and physical data about a volume, when passed a dirHandle pointing to the volume. The function returns the volume's name, the number of blocks on the volume, the number of sectors per block, the number of unused blocks on the volume, the number of directory slots on the volume, the number of unused directory slots, and a flag indicating whether the volume is removable. To obtain a directory handle, the application must call NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle.

The NWGetVolUsage function also returns information about what is available, and in use, on a certain volume. The volNum parameter identifies the volume on the file server's Volume table, which contains information about each volume on the file server.

Both functions pass three parameters with the exception that one passes dirHandle and the other passes volNum.

NWGetVolInfoWithHandle(serverConnID, dirHandle, &volUsage)

NWGetVolUsage(serverConnID, volNum, &volUsage)

Trustees

Trustees can be assigned to directories and files. The File Service APIs allow you to add new trustees, delete existing trustees, get trustee information, and modify trustee assignments. Each task is described below.

Deleting Trustees

The NWDeleteTrustee function revokes a trustee's rights in a specific directory and removes the trustee from a directory's trustee list. To delete a trustee, the requesting client must have Access Control rights to the directory or file.

Deleting the explicit assignment of an object's trustee in a directory is not the same as assigning that object no rights in the directory. If no rights are assigned in a directory, the object inherits the same rights it has in the parent directory.

This function passes two parameters as shown below. The trusteeObjectID can be obtained by calling NWGetObjectID.

NWDeleteTrustee(&path, trusteeObjectID)

Getting Rights Information

Three calls get information related to rights assignments, such as effective rights, trustees, or the directory paths that trustees have rights to.

The NWGetEffectiveRights function returns a client's effective rights in the specified directory. This function passes two parameters.

NWGetEffectiveRights(&path, &effectiveRightsMask)

The requesting workstation's effective rights are determined by ANDing the inherited rights mask of the directory with the client's current trustee rights. For detailed explanations of how effective rights are derived, see the section, "Effective Rights," in this chapter.

The NWGetEntrysTrustees function returns trustee information such as trustee objectIDs and their associated rights. The number of trustees returned is specified by the application. The numberOfEntries returns the actual number of trustees found by the call.

This call passes four parameters.

```
NWGetEntrysTrustees( &path, &numberOfEntries, trusteeRights,
    maxListElements )
```

The NWScanTrusteePaths function returns the directory paths to which an object has trustee rights. This function is used iteratively to determine all the trustee directory paths of a bindery object and their corresponding access masks.

This call passes six parameters as shown below.

```
NWScanTrusteePaths( serverConnID, objectID, volNum, &entryID,
    &trusteeAccessRights, directoryPath )
```

Setting Trustees

The NWSetTrustee function creates a trustee or changes a current trustee's trustee rights to a directory. To take rights away from a trustee, simply pass a trusteeRightsMask without those bits set. This function passes three parameters as can been seen in the entry below.

```
NWSetTrustee( &path, trusteeObjectID, trusteeRightsMask )
```

The trusteeRightsMask can be obtained by ORing together all of the desired trustee rights.

Salvageable Files

When files are deleted under NetWare, they are actually saved in the buffer. The deleted files can be purged, restored, or viewed by using the following calls.

Note: Your NetWare for AViiON Systems version may not support Salvaging Files and Purging Files. Refer to the release notice accompanying your shipment for specific restrictions.

Scanning Salvageable Files

The NWScanSalvageableFiles function returns information on deleted, but salvageable, files. This call is only valid for NetWare version 3.x. It passes four parameters.

```
NWScanSalvageableFiles( serverConnID, dirHandle, &entryID,
    &salvageableInfo )
```

Purging Files

The NWPurgeSalvageableFile function permanently deletes files that have been erased but are still recoverable. This function frees up the disk space used by deleted, but still recoverable files. It is only valid on NetWare version 3.x.

The NWPurgeSalvageableFile function passes two parameters: a pointer to the structure path and entryID. The entryID parameter can be obtained by calling NWScanSalvageableFiles.

Recovering Files

The NWRecoverSalvageableFile function restores a deleted, but salvageable file. This call can only restore files that were deleted by the last call to NWDeleteFile, and cannot recover files that were marked for deletion prior to a call to NWPurgeSalvageableFile.

To call this function, you should first use the NWScanSalvageableFiles until you find a file you want to salvage. Then, call the NWRecoverSalvageableFile passing in the entryID from NWScanSalvageableFiles corresponding to the desired file. This function is only valid on NetWare version 3.x.

This call passes three parameters.

```
NWRecoverSalvageableFile( &path, entryID, newFileName )
```

If an application running on a different workstation creates a file with the same name as the deleted file, the function renames the recovered file, replacing the last two characters of the file extension with 00. For example, FILE.T00 replaces FILE.TXT.

End of Chapter

Chapter 5
Path Services

The Path Services calls enable an application program to allocate and deallocate directory handles.

Tables Accessed By Path Services

File servers and workstations maintain several tables that are used by Path Service APIs. They are the following: Directory Table and Directory Handle Table.

Directory Table

To record information about directories, a file server maintains a Directory Table. Entries in the Directory Table define a file server's directory structure; all Directory Services calls access the Directory Table. The Directory Table contains 3 kinds of entries: directory nodes, file nodes, and trustee nodes.

A directory node includes the following information about a directory: directory name, attribute byte, maximum rights mask, creation date, creator's object ID, a link to the parent directory, and a link to a trustee node (if one exists).

A file node includes the following information about a file: file name, attribute byte, file size, creation date, last-accessed date, last-updated date and time, last archive date and time, the file owner's object ID, and a link to a directory.

A trustee node includes the following information: the object IDs of one to five trustees of a directory linked to the trustee node, one to five corresponding trustee rights masks, a link to a directory, and a link to the next trustee node (if one exists).

Directory Handle Table

The file server maintains a directory handle table for each workstation logged into it. The directory handle table has 256 entries (0x00...0xFF), each of which can be set to point to a volume or directory path. When a workstation requests a directory handle, the file server enters the volume number and directory entry number for the specified directory into the directory handle table for the requesting workstation. Applications running on the workstation can then refer to a directory using a directory handle, which is actually an index into the directory handle table.

Directory handles are discussed in greater detail in "Directory Handles" later in this chapter.

Directory Handles

Once a directory handle is set, you can use the directory handle to specify a volume or a directory path on the file server. NetWare APIs that refer to directories permit you to specify a directory path in three different ways.

- Allocate a directory handle to point to the target directory.
- Use both a directory handle and a complementary directory path to specify the complete path. The directory handle must lead part of the way to the target directory, and the directory path must lead the rest of the way.
- Specify the entire directory path.

When an application begins executing, the application must allocate all the directory handles that it needs.

The path services uses the NWPath_t structure below to specify the location of NetWare file or directory.

NWPath_t Structure

```
typedef struct {
    NWDirHandle_ts   dirHandle;
    uint16           serverConnID;
    char             *pathName;
} NWPath_t;
```

The **dirHandle** field represents the directory handle allocated by the client pointing to a particular place in the directory structure. The dirHandle is specific to the file server connection (serverConnID) and can only be used to access directories or files on that file server.

The **serverConnID** field represents the file server connection which contains the file system being accessed.

The **pathName** field is a pointer which points to a character string which the client must allocate and fill in with a path name.

The NWPath_t structure can be used in one of three ways:

1. The application can pass a 0 in the dirHandle field and then pass a full path (of the target directory or file) in the pathName field. In the following example, the path to be accessed is the SYS:APPS/WP directory on file server connection 0 (first file server attached to).

```
NWPath_t          path;
char         fullPath[NWMAX_DIR_PATH_LENGTH];

path.serverConnID = 0;
path.dirHandle = 0;
strcpy( fullPath, "SYS:APPS/WP" );
path.pathName = fullPath;
```

2. The application can pass a previously allocated directory handle in the dirHandle field (see NWAllocTemporaryDirHandle or NWAllocPermanentDirHandle) and then can pass a path in the pathName field which is relative to the directory that the dirHandle points to. In the following example, the path to be accessed is the SYS:APPS/WP directory on file server connection 0 (first file server attached to). A dirHandle of 3 has already been allocated for SYS:APPS.

```
NWPath_t      path;
char          partialPath[NWMAX_DIR_PATH_LENGTH];

path.serverConnID = 0;
path.dirHandle = 3;
strcpy( partialPath, "WP" );
path.pathName = partialPath;
```

3. The application can allocate a directory handle to point to the target directory. Note that the pathName is null when the dirHandle being passed already references the full path. In the following example, the path to be accessed is the SYS:APPS/WP directory on file server connection 0 (first file server attached to). A dirHandle of 4 has already been allocated for SYS:APPS/WP.

```
NWPath_t      path;
char          noPath[NWMAX_DIR_PATH_LENGTH];

path.serverConnID = 0;
path.dirHandle = 4;
strcpy( noPath, "" );
path.pathName = noPath;
```

Path Services provides six calls to allow the manipulation of directory handles:

- NWAllocPermanentDirHandle
- NWAllocTemporaryDirHandle
- NWDeallocateDirHandle
- NWGetDirPath
- NWParseFullPath
- NWSetDirHandle

The NWAllocPermanentDirHandle function allows an application to assign a directory handle and permanently maps a workstation drive to a network directory.

The NWAllocTemporaryDirHandle function temporarily assigns a directory handle and maps a workstation drive to a network directory.

The NWDeallocateDirHandle function deallocates a previously allocated directory handle. However, the file server automatically deallocates directory handles under the following situations:

- Temporary directory handles are deallocated when the application loses its connection (logs out).
- Permanent directory handles are deallocated when another permanent handle is allocated to the same path.

The full path that a directory handle points to can be seen using the NWGetDirPath function call. To further see the different parts of a full path, the NWParseFullPath is used.

The NWSetDirHandle function assigns a directory handle to a file server directory path (relative to an existing directory handle).

Permanent and Temporary Directory Handles

Temporary directory handles exist until the application that allocated them exits or calls the EndOfJob function. The mapping of a workstation drive letter to a permanent directory handle continues after the application that created the mapping exits. Temporary directory handles also differ from permanent directory handles in that several temporary directory handles can be assigned the same drive letter, but still be separate handles mapped to different directory paths.

Applications allocate permanent directory handles by calling NWAllocPermanentDirHandle, which takes the same parameters as NWAllocTemporaryDirHandle. Permanent directory handles are not automatically deallocated when the application that allocated them exits. The NWDeallocateDirHandle function is provided to deallocate directory handles. Even though NetWare automatically deallocates temporary directory handles when the application that allocated them exits, some application developers prefer to explicitly deallocate temporary directory handles as a matter of programming style.

File Paths

When specifying a file to a file services call, an application can use either a full or a partial file path. A full file path has the following format:

Volume:\Directory\...\Directory\File

Because a full file path completely specifies a file, file services calls ignore the value of their directory handle parameter when an application passes a full file path.

A partial file path includes a file name and optionally one or more antecedent directory names. The file service routine then combines the directory handle setting and the partial file path to obtain a full file specification. For example, suppose an application calls a file services routine and passes a directory handle mapped to the directory WORK:\HOME\MARY and a file path parameter containing the partial file path ACCTS\ACTIVE\ZZYZX.DAT. The routine would use the directory handle and partial file path to identify the file WORK:\HOME\MARY\ ACCTS\ACTIVE\ZZYZX.DAT.

A full path, including a server name has the following format:

Server/Volume:\Directory\...\Directory\File

The NWParseFullPath function call uses the full path, including server name.

End of Chapter

Chapter 6
Queue Services

Queue Services make NetWare's Queue Management System (QMS) available to developers. To explain queue services, this chapter is divided into the following sections:

- Why Use QMS?
- Queues and the Bindery
- The Queue Process
- Using Queue Services

Why Use QMS?

Among the methods that developers can use to distribute processes on the network, QMS is perhaps the simplest and most direct. Although not a suitable solution for all situations, QMS offers some advantages over other methods.

Applications that need to do the following may want to consider using QMS:

- Exercise control over the flow and execution of large workloads
- Define specifications and protocols for processing jobs
- Provide broad safeguards to protect job information and network data

Control

Generally, QMS is suited to applications that are dealing with large workloads and need the flexibility and control inherent in the queuing process. Time and efficiency are somewhat less important to these applications than are security and reliability.

For example, an archive server is a good candidate for QMS. An archive server cares less about speed than about a dependable and secure operating environment. In addition, archive jobs may need to be sorted and managed according to size, priority, kind, and other properties. QMS offers the regulatory features these tasks require.

Flexibility

Flexibility is another one of QMS's advantages. Although every queue must adhere to QMS's standard structure, QMS allows applications to define their own specifications within the QMS format. Thus, queues can transmit information that is specific to a specialized service. For example, a print server could use QMS to define its own protocol for handling diverse printing formats.

Security

QMS is based on NetWare's Bindery and takes advantage of the Bindery's extensive security structure. Along with NetWare's directory security, the Bindery's organization clearly defines the relationship among queues, users, and queue servers. An application can ensure that only qualified users work with network data, whether it is placed temporarily in a queue or stored permanently on network disks.

In addition to the advantages listed so far, QMS is relatively easy for developers to use. Since QMS exploits NetWare's existing design and security features, it avoids possible conflicts with other NetWare services. Developers who are already familiar with NetWare's Bindery should find QMS's design familiar and accessible. In addition, Queue Services contain an extensive group of function calls that help make QMS simple and convenient to manage.

Queues and the Bindery

Queues form the basis of QMS. Simply put, a queue is a group of jobs waiting to be serviced. On one end of the queue are users, who submit jobs, and at the other end are queue servers that process the jobs. For the most part, jobs proceed sequentially through the queue, those arriving first being serviced before those that come after.

Queues are objects in a file server's bindery. Defining a queue as a bindery object allows QMS to guarantee that only authorized users will access and modify the queue. By defining users, operators, and servers as properties of the queue, QMS provides extensive control over how the queue is used and who may use it. For more information about the structure of the Bindery, see Chapter 2.

The Queue Object

As with any bindery object, a queue has a name, type, ID number, and security status. The queue's name and ID number identify it uniquely among queues in a file server's bindery. Particularly on an internetwork, it is important to note that a queue exists in the bindery of a specific file server.

QMS limits each queue to a single type of service. When an application creates a queue, it must define the type of jobs that queue intends to hold. Some queue types are designated by Novell. For example, a print queue is defined as NWOT_PRINT_QUEUE and will only accept jobs that correspond to that type.

When QMS creates a queue, the queue is assigned the security levels NWBS_ANY_READ (read anyone) and NWBS_SUPER_WRITE (write supervisor). The NWBS_ANY_READ level allows users to scan the bindery for a list of queues of a particular type. On the other hand, the NWBS_SUPER_WRITE level ensures that only the supervisor will be able to modify the queue's properties.

The features of a queue object are given in Table 6-1. The queue properties are described below.

Table 6-1 The Queue Object

Feature	Explanation
Name	Assigned by the application
ID number	Assigned by QMS
Type	A number identifying the queue's work
Status	NWBS_ANY_READ | NWBS_SUPER_WRITE
Properties	Q_DIRECTORY Q_USERS Q_OPERATORS Q_SERVERS

Queue Properties

QMS attaches four properties to the queue object in the bindery: Q_DIRECTORY, Q_USERS, Q_SERVERS, and Q_OPERATORS. These properties let an application control access to the queue through normal bindery routines.

The queue properties Q_USERS, Q_OPERATORS, and Q_SERVERS define the role of queues and their relationship with other QMS components. In most case, these properties are sufficient to accommodate a queue server's needs, although an application may define additional queue properties, as well. (For example, a queue could define as a property the type of hardware device that the queue is associated with--printer, plotter, tape, etc.)

The Q_DIRECTORY Property

The first property, Q_DIRECTORY, has a security level of NWBS_SUPER_READ | NWBS_SUPER_WRITE (read supervisor and write supervisor). The other three properties are NWBS_ANY_READ | NWBS_SUPER_READ (read anyone and write supervisor). Thus, only the supervisor can see or modify the value of Q_DIRECTORY, but any logged in user can see the value of Q_USERS, Q_SERVERS, and Q_OPERATORS.

When an application creates a queue, it must specify a directory path. QMS will use this path to create a queue directory that holds the system files and job files related to queue entries. The directory path SYS:SYSTEM is commonly used for the queue directory; however, you may specify any path.

QMS names the queue directory with queue's ID number. The complete directory path, including the queue directory, is assigned to the Q_DIRECTORY property. The path string specified by the application cannot exceed NWMAX_PROPERTY_VALUE_LENGTH (128 characters, including the null terminator).

Q_DIRECTORY is an item property; each queue has only one.

Table 6-2 Example of a Queue Directory

Feature	Explanation
Path	SYS:\SYSTEM
Queue ID	015D03
Queue Directory	SYS:\SYSTEM\015D03

The Q_USERS Property

QMS makes queues available to users through the Q_USERS property. This set property is a list of all the users who may place a job in a queue. The list can contain both user names and group names.

If a user is the security equivalent of any object listed in the Q_USERS property, that user has access to the queue. Likewise, when a group name is added to the Q_USERS property, all users in the group have access to the queue. Since the Q_USERS property is NWBS_SUPER_WRITE, only the supervisor can add or remove users from a queue. However, to use the queue, the supervisor must also be added to the Q_USERS property, just like any other object. If the supervisor does not want to place any restrictions on the queue, the group EVERYONE can be added to the Q_USERS property.

The Q_SERVERS Property

For a queue to be serviced, a value must be assigned to the queue's Q_SERVERS property. This property holds a list of all the queue servers that can handle jobs placed in the queue. If an application does not assign any queue servers to the Q_SERVERS property, jobs may be placed in the queue, but they will not be serviced.

Before an application can assign a queue server to a queue, the queue server must exist as an object on the file server where the queue resides. As a bindery object, a queue server can log in to the file server and search for queues. QMS checks a queue server's login name and password against the servers listed in a queue's Q_SERVER property to determine if the queue server is authorized to service the queue.

When an application creates a queue server as a bindery object, it must specify the queue server's object type. As with queue types, Novell has designated object types for queue servers. The following object types are designated as queue servers:

Object	Type
Print server	NWOT_PRINT_SERVER
Job server	NWOT_JOB_SERVER
Archive server	NWOT_ARCHIVE_SERVER

If you need an object type that is not defined, contact Novell's API Consulting Group.

An application must also assign an account balance to the queue server.

The Q_OPERATORS Property

QMS attaches a Q_OPERATORS property to each queue, which contains a list of all users who are authorized to manage the queue. An application must assign values to a the Q_OPERATORS property before the queue can be managed. Operators can modify the specifications for a particular job, as well as manipulate the order of jobs in the queue. Operators can also set the status of a queue, suspending the operation of the queue as needed.

As with the Q_USERS property, user or group names can be added to the Q_OPERATORS property. In other words, queue operators must exist in a file server's bindery as user and group objects. In order for the object SUPERVISOR to perform operator tasks, it must also be added as an object to the Q_OPERATORS property

Supervisors and QMS

Developers should keep in mind the different security levels involved in a QMS application. An application needs supervisor equivalence to install and configure a queue server, since only supervisors can perform the bindery operation related to QMS.

However, queue users and queue operators need no special rights to exercise their privileges. Any user assigned to a queue's Q_USERS property can submit jobs, and any user assigned to a queue's Q_OPERATORS property can manage the operation of the queue.

The Queue Process

When a queue is created in the bindery, QMS automatically assigns it a Q_DIRECTORY property containing the queue directory path. Within this directory, QMS creates two system files that it uses to manage the queue. As jobs are submitted to the queue, temporary queue files are also placed in the queue directory.

QMS Files

Queue files are named with the characters Q$ followed by the last four digits of the queue's ID number. One file maintains a list of queue servers currently attached to the queue and has the extension .SRV. The other file maintains information about the jobs in the queue and has the extension .SYS. Both files are flagged HIDDEN when created.

A job is submitted to a queue as a queue file. Queue files use the same naming conventions as the queue system files, only each extension is a three digit number. The extension numbers of queue files reflect the sequence of queue jobs as they are created (.001, .002, .003, etc.).

When a queue file is created, a queue job structure (NWQueueJobStruct_t) is appended to the beginning of the file. The queue job structure takes up the first 255 bytes of the queue file. QMS uses this information to process the job. As jobs are processed, both the queue file and its reference within the .SYS file are deleted.

The queue file itself can contain any data needed for completing the job. If the queue supplies printing services, the queue file could contain the data to be printed. Or, if the queue supplies compiling services, the queue file might contain a compilation makefile. Developers are also free to use the queue file for transmitting their own protocol information.

The NWQueueJobStruct_t Structure

All the information QMS requires to process a job is placed in the NWQueueJobStruct_t. Some of this information must be supplied by the application that creates the queue job; the rest is supplied by QMS. After the job is in the queue, the queue operator or the user who submitted the job can modify the information that indicates how to process the job.

NWQueueJobStruct_t Fields

Table 6-3 lists all of the fields in the NWQueueJobStruct_t Structure. A description of each field follows the table.

Table 6-3 NWQueueJobStruct_t Structure

Field	Type
clientStation	uint8
clientTask	uint8
clientID	uint32
targetServerID	uint32
targetExecutionTime	uint8[NWMAX_QUEUE_JOB_TIME_SIZE]
jobEntryTime	uint8[NWMAX_QUEUE_JOB_TIME_SIZE]
jobNumber	uint16
jobType	uint16
jobPosition	uint8
jobControlFlags	uint8
jobFileName	uint8[NWMAX_JOB_FILE_NAME_LENGTH]
jobFileHandle	NWFileHandle_ta
servicingServerStation	uint8
servicingServerTaskNumber	uint8
servicingServerIDNumber	uint32
jobDescription	uint8[NWMAX_JOB_DESCRIPTION_LENGTH]
queueRecord	NWClientRecord_ta

QMS automatically fills in all the fields in the structure except the following which must be filled in by the application:

targetServerID
targetExecutionTime
jobType
jobControlFlags
jobDescription
queueRecord

clientStation. This field contains the number of the station that placed the job in the queue.

clientTask. This field contains the number of the task the station was performing when it placed the job in the queue.

clientID. This field contains the user ID number of the station that placed the job in the queue.

targetServerID. This field contains the ID number of the queue server that is requested to service the job. A value of FF FF FF FF (4 bytes of 0xFF) indicates that any server may service the job. Otherwise, this field contains a queue server's ID number.

targetExecutionTime. This field contains the earliest time the client wants the job to be serviced. If this field is set to FF FF FF FF FF FF (6 bytes of 0xFF), the job will be serviced at the first opportunity.

This 6 byte field is divided as follows.

First byte	Year	0 - 99 (91 for 1991)
Second byte	Month	1 - 12
Third byte	Day	1 - 31
Fourth byte	Hour	0 - 23
Fifth byte	Minute	0 - 59
Sixth byte	Second	0 - 59

jobEntryTime. This field contains the time that the job was placed in the queue. The time is taken from the system clock on the file server where the queue is found. This field is a 6 byte field. See "targetExecutionTime" above for an explanation of the bytes.

jobNumber. This field contains the number that QMS assigns to the queue entry when the job is placed in the queue. A client should use this number when referring to the job.

jobType. This field contains a number that identifies the job by type. This field can contain any additional information that the client needs to transmit to the queue server. Applications are free to define and interpret this field as needed. Developers should avoid assigning -1 to a job type. If the queue server does not use the field, set the field to 0x00.

jobPosition. This field contains the position of the job in the queue. The entry at the head of the queue is number 1, the next entry is 2, and so on. As jobs are removed from the queue, the position numbers are updated to reflect the new position of the jobs that remain.

jobControlFlags. This field contains flag bits indicating the current status of the job. The bits are defined in Table 6-4.

Table 6-4 Job Control Flags

Flag	Explanation
NWCF_SERVICE_AUTO_START	Setting this bit allows a job to go in the queue automatically if the connection is broken between the station submitting the job and the file server where the queue resides. Clearing this bit automatically removes a job from the queue if the connection breaks and the client has not yet released the job.
NWCF_SERVICE_RESTART	Setting this bit allows a job to remain in a queue in the event of a service failure. The queue server will attempt to service the job when service to the queue resumes. Clearing this bit allows the job to be removed from the queue if it has not been serviced successfully.
NWCF_ENTRY_OPEN	This bit is for QMS's internal management. It remains set as long as the client has not released the queue file to the queue. When the client has released the job, QMS clears the bit.
NWCF_USER_HOLD	Setting this bit places the job on hold. The job will continue to advance in the queue, but it will not be serviced until this bit is cleared.
NWCF_OPERATOR_HOLD	Setting this bit places the job on hold. The job will continue to advance in the queue, but it will not be serviced until this bit is cleared.

jobFileName. This field contains the name of the queue file created to process the job.

jobFileHandle. This field contains the file handle to the queue file that has been created in the queue. The handle can be used to write additional information to the file.

servicingServerStation. This field contains the task number of the queue server servicing a job. When no server is servicing a job, the value of this field is left undefined.

servicingServerIDNumber. This field contains the object ID of the queue server servicing a job. When no server is servicing the job, this field is set to zero. By testing this field, an application can determine whether a job is being serviced.

jobDescription. This field contains a null-terminated ASCII string. Applications can use this field to help identify the type or purpose of the job.

queueRecord. This field can contain any additional information that the client needs to transmit to the queue server. Applications are free to define and interpret this field as needed. Any values placed in this field will not be affected by QMS.

Managing the NWQueueJobStruct_t Structure

The NWQueueJobStruct_t structure plays several distinct roles in the queuing process. QMS, queue servers, clients and operators all depend on certain fields to transmit information at various points in the job process.

1. First, the following fields must be completed before the job can be placed in the queue.

   targetServerID
   targetExecutionTime
   jobType
   jobControlFlags
   jobDescription
   queueRecord

2. Next, when a job is placed in the queue, QMS records information in the following fields.

   clientStation
   clientTask
   clientID
   jobEntryTime
   jobNumber
   jobPosition
   jobControlFlags [NWCF_ENTRY_OPEN]
   jobFileName
   jobFileHandle

3. Once the job is in the queue, the user who submitted the job or the queue operator can read and modify information in the following fields:

   targetServerID
   targetExecutionTime
   jobType
   jobPosition
   jobControlFlags
   jobDescription
   queueRecord

4. When a server attaches to the queue, QMS checks the following fields to determine if the server is authorized to handle the job.

   targetServerID
   targetExecutionTime
   servicingServerIDNumber
   jobType
   jobControlFlags

   The server cannot service the job if one of the following is set:

   NWCF_OPERATOR_HOLD
   NWCF_USER_HOLD
   NWCF_ENTRY_OPEN

5. Finally, when QMS releases a job to a queue server, the following fields are filled in:

   servicingServerStation
   servicingServerTaskNumber
   servicingServerIDNumber

Using Queue Services

Queue services has five areas of management:

- Creating queues in the bindery
- Submitting jobs to queues
- Monitoring and controlling the status of jobs and queues
- Creating queue servers in the bindery
- Servicing entries in queues as a queue server

Not all applications need to manage all five areas. For example, a simple word processing application could supply only the code to submit the job into a print queue. A more sophisticated application could also allow users to monitor and control their jobs in the queue. However, a print server would mainly be concerned with the last area, servicing entries in a print queue.

Each area of management is described below.

Creating Queues in the Bindery

Queues are created with NWCreateQueue. The client that creates the queue needs to be logged in with supervisor equivalency.

The NWCreateQueue creates the queue as a bindery object and as a directory. Typical bindery types for queues are listed below:

Object	Type
Print Queue	NWOT_PRINT_QUEUE
Archive Queue	NWOT_ARCHIVE_QUEUE
Job Queue	NWOT_JOB_QUEUE

It also creates four bindery property types and assigns them to the queue:

Q_DIRECTORY
Q_SERVERS
Q_OPERATORS
Q_USERS

The Q_DIRECTORY property stores the path information for the queue directory. Usually a standard naming convention is used for the directory. For example, the print queue directories created with PCONSOLE are subdirectories to SYS:SYSTEM, have a .NPQ extension, and are named by giving them the character value of their hexadecimal object ID. The Q_DIRECTORY property is an item property.

The other three properties are set properties and allow you to assign bindery objects to them.

- The Q_SERVERS property stores the object IDs of the servers that have been added to this property. These servers, and only these servers, can service the queue. If no servers are assigned to the queue, no jobs will be serviced. The queue server must be created as a bindery object before the queue server can be assigned to this property. See "Creating Queue Servers in the Bindery," in this chapter.
- The Q_OPERATORS property stores the object IDs of the users or groups that can perform maintenance tasks on the queues. These objects are able to delete all jobs in the queue, rearrange the order of the jobs in the queue, change job servicing information, and control the submission of new jobs. Usually SUPERVISOR is added to this group.
- The Q_USERS property stores the object IDs of the users or groups that have permission to use the queue. Each object can delete its own jobs or change the job servicing information. If no objects are assigned to the queue as queue users, no one can submit a job to the queue. Usually the group EVERYONE is added to this group.

Use NWAddObjectToSet to add bindery objects to the properties. Use NWDeleteObjectFromSet to remove a bindery object from the properties.

Use NWCreateProperty to create additional bindery properties for the queue. If you create a set property, use NWAddObjectToSet to add objects to the property. If you create an item property, use NWWritePropertyValue to write the value to the property.

Use NWDestroyQueue to delete a queue from the bindery.

Submitting Jobs to Queues

For a client to submit a job to a queue, the client must have its object ID (or the object ID of a group that client belongs to) in the Q_USERS property.

To submit a job to the queue, complete the following steps:

1. Assign values to the necessary fields in the NWQueueJobStruct_t Structure. (See "NWCreateQueueFile" in the Reference Guide.)
2. If the queue server uses the queueRecord field, assign a value to the queueRecord field.
   - If the job is a NetWare print job, use NWConvertPrintStructToQueueStruct.
   - If the job is not a NetWare print job, use the queue server's function to fill in the queueRecord field.
3. Create a file in the queue. Use NWCreateQueueFile.
4. Write the file to the queue. Use NWWriteFile. Use the file handle that was created with NWCreateQueueFile.
5. Close the file. Use NWCloseFileAndStartQueueJob. If an error occurs, you can use NWCloseFileAndAbortQueueJob which closes the source file and removes the job from the queue.

The job is now ready to be serviced.

Monitoring and controlling the status of jobs and queues

The ability to monitor and control jobs in the queue varies.

- Clients which have been assigned to the Q_OPERATOR property can control the queue and the jobs.
- Clients which have been assigned to the Q_USERS property can control their own jobs and as users, they can view information about the queue and other user's jobs in the queue.
- Clients which have not been added to either property cannot view the queue status or queue job information.
- Queue servers which have been added to Q_SERVERS property can read and service jobs in the queue.

Table 6-5 Queue Controls

Task	API	Client
Delete a job	NWRemoveJobFromQueue	Owner Operator
View job information	NWReadQueueJobEntry*	User Operator Server
Change job information	NWChangeQueueJobEntry	Owner Operator Server
List jobs in the queue	NWGetQueueJobList	User Operator Server
Change servicing order	NWChangeQueueJobPosition	Operator
View queue status flags	NWReadQueueCurrentStatus	User Server
Set queue status flags	NWSetQueueCurrentStatus	Operator
View queue server flags	NWReadQueueServerCurrentStatus	User
Set queue server flags	NWSetQueueServerCurrentStatus	Operator

* If the job is a NetWare print job, use NWConvertQueueStructToPrintStruct before reading the job. If the job is not a NetWare print job and the queue server required a function to change the job into a queue job, use the queue server's function to return the job to its original form.

Creating Queue Servers in the Bindery

To create a queue server, follow the steps outlined below.

1. Create the queue server as a bindery object. The client that creates the queue server needs to be logged in as supervisor equivalent. Use NWCreateObject to create the queue server. Assign the object an appropriate object type. Typical types for queue servers are listed below:

Object	Type
Job Server	NWOT_JOB_SERVER
Print Server	NWOT_PRINT_SERVER
Archive Server	NWOT_ARCHIVE_SERVER

2. Get the object ID of the queue server. Use NWGetObjectID. (The object ID number must be passed with some of the other API calls.)

3. Create a PASSWORD property for the queue server and assign the property a value. Use NWChangeObjectPassword.

4. Create properties for the queue server. Use NWCreateProperty. For example, the print server creates two properties: PS_USERS and PS_OPERATORS. Both of these properties are of the type set and are given the access level of NWBS_LOGGED_READ, NWBS_LOGGED_WRITE, and NWBS_OBJECT_WRITE.

5. Check for accounting on the file server. Use NWScanProperty and look for a file server property of ACCOUNT_SERVERS. If this property exists, create an ACCOUNT_BALANCE property for the queue server and write an ACCOUNT_BALANCE value to the property. The ACCOUNT_BALANCE property is of type item so use NWWritePropertyValue.

6. If you need a directory to store information in, create a directory for the queue server. Use NWCreateDir. Assign the queue server to be a trustee of the directory. Use NWSetTrustee.

7. Add objects to the properties created in Step 4. Usually user SUPERVISOR will be added to an operator property and group EVERYONE is added to a user property. For set properties, use NWAddObjectToSet. For item properties, use NWWritePropertyValue.

To delete a queue server from a file server, use the following APIs:

- Use NWDeleteObject to delete a queue server from the bindery.
- Use NWDeleteDir to delete any directories created for the queue server.

Servicing Entries in the Queue

To service jobs in a queue, the queue server needs to perform the following tasks.

1. Log the queue server into the file server. Use NWAttachToServerPlatform and NWLoginToServerPlatform.

2. Attach the queue server to the queue. Use NWAttachQueueServerToQueue. The queue server must first be a member of the queue's Q_SERVERS property. See "Creating Queues in the Bindery" in this chapter.

3. Get the current status of the queue. Use NWReadQueueCurrentStatus.

4. Check to see if there are jobs to be serviced. Use NWGetQueueJobList.

5. Find the next job that needs to be serviced. Use NWReadQueueJobEntry. If the queue server needs to know the size of the job, use NWGetQueueJobSize.

6. Select the job for servicing. Use NWServiceQueueJob.

7. Read the job. Use NWReadFile.

8. Process the job. This varies with the queue server. A print server would send the file to the printer with the appropriate printing codes. An archive server would send the file to a backup device with the appropriate formatting codes.

   The process could also involve having the queue server assume the rights of the client. For example, an archive server could assume the rights of the client before restoring files to enforce the following:

   - The clients could restore only files that they had the appropriate rights to.
   - The clients could restore files only to directories that they had the create right in.
   - The clients could be assigned as the owner of the restored file.

   You should use NWChangeToClientRights before processing the job and then use NWRestoreQueueServerRights when the job is finished.

9. Remove the job from the queue. Use NWFinishServicingQueueJob.

10. Repeat Step 4 through Step 9 for additional jobs in the queue.

NWDetachQueueServerFromQueue should be used before bringing down the queue server.

NWAbortServicingQueueJob can be used to interrupt the servicing of a job.

End of Chapter

Chapter 7
Server Platform Services

This chapter discusses Server Platform (File Server) services. The Server Platform calls are closely related to the Connection Services calls. Server Platform services allow a client to enable and disable file server login, and return information and statistics about file server configurations and activity. The server information returns in a structure form as defined in Appendix A of the *NetWare® for AViiON® Series Systems C Interface Reference Guide.* Most of these calls require supervisory access privileges.

Function Calls

The Server Platform calls provide two major types of services: 1) Disable and enable servers and 2) Get and set server information. Listed below are the calls and their services.

Disable and Enable Servers	Get and Set Server Information
NWDisableServerPlatformLogin	NWGetDiskUtilization
NWDownServerPlatform	NWGetServerPlatformDateAndTime
NWEnableServerPlatformLogin	NWGetServerPlatformInformation
	NWGetServerPlatformName
	NWGetServerPlatformDescriptionStrings
	NWGetServerPlatformLoginStatus
	NWIsNetWare386
	NWSetServerPlatformDateAndTime

Date and Time

The information on date and time on the server is not the same as those defined in the file system services. Server date and time information are stored in a structure while file system's date and time information are stored in 4 bytes. Table 7-1 lists the fields in the NWServerPlatformDateAndTime_t Structure. A description of each field follows the table.

Table 7-1 NWServerPlatformDateAndTime_t Structure

Field	Type
year	uint8
month	uint8
day	uint8
hour	uint8
minute	uint8
second	uint8
dayOfWeek	uint8

year. The year field is passed in a value from 0 through 99. For example, 91 becomes 1991, and 10 becomes 2010.

month. The month field passes in a value from 1 through 12.

day. The day field passes in a value from 1 through 31.

hour. The hour field passes in a value from 0 through 23. Noon is 12.

minute. The minute field passes in a value from 0 through 59.

second. The second field passes in a value from 0 through 59.

dayOfWeek. The dayOfWeek field passes in a value from 0 through 6. Sunday is 0.

Boolean Functions

NWIsNetWare386 is the only server call to return 1 if success or 0 if fails. This call returns a boolean rather than an integer type.

Disabling Logins

Disabling the login does not allow users to login. If the application disables the login and then loses its connection (or logout), the server will have to be rebooted. For practical reasons, the application should disable the login, perform intended tasks or operations on the server, and then enable the login.

To see whether the login is disabled, call NWGetServerPlatformLoginStatus.

Getting and Setting Server Information

The types of server information available to an application are listed below. The list also includes how each type of information is set.

Types of Server Information	How Is the Information Set
Server's Date and Time	Calling NWSetServerPlatformDateAndTime
Server Platform Information	When the server is configured
Server Name	Set at installation
Server Description Strings	Internal to NetWare
Server Login Status	Call NWDisableServerPlatformLogin or NWEnableServerPlatformLogin

End of Chapter

Chapter 8
Synchronization Services

This chapter explains the NetWare services that control file and record sharing. Developers can use these calls to manage file access among users on the network. This chapter includes the following topics:

- An Introduction to File Sharing
- Locking Files and File Sets
- Locking Records
- Semaphores
- File-sharing Strategies

An Introduction to File Sharing

One of the major aims of a NetWare network is to allow many users to easily access data that is kept on common storage devices. The file server controls access to these storage devices, retrieving and writing data as requested by workstations. Such files can be considered network files, in contrast to local files that are stored on a workstation's local disk drives and are accessed only by that workstation.

Through the file server, workstations not only have access to the same network files, but they have access to the same network files at the same time. Several workstations can open the same file, retrieve its contents into workstation memory, make changes, and write the file back to storage.

So long as network files are intended to be read only, no conflict should occur between workstations as they retrieve files and read their contents. However, when network files need to be altered and updated, the potential for conflict becomes very great.

If even two workstations, independent of one another, are changing the contents of a file at the same time, the results can be disastrous. Since files are copied into workstation memory as they are read, both workstations are only dealing with a "working copy" of the file and must write any changes they make back to the disk. As both workstations update the network file, neither one is aware of the changes the other is making.

Synchronization Services are designed to help applications avoid the sort of file-sharing conflicts described above. These services are based on the concept of data locking. Before an application allows a user to modify a file, the file is locked so that only that user has access to it. As soon as the modifications are made, the application releases the file, allowing other users to modify it.

In situations that involve many interdependent files, the potential for problems is compounded. Groups of files may have to be locked together until a single modification is carried through to the end. In the meantime, the application may permit other users to read the locked files, so long as the do not attempt to change anything.

Data locking reduces the likelihood of two workstations attempting to update the same file simultaneously, but it is only a partial solution. Data locking does not solve the problem of one station being unaware of the changes another station has introduced into a file.

For example, suppose that workstation #1 locks a file and modifies it while workstation #2 is reading the file. When workstation #1 releases the file, workstation #2 is free to write to it. But workstation #2 is working on an instance of the file that does not include the current changes made by workstation #1. From banking records to travel reservations to inventory control, the scenarios for a catastrophic error resulting from this arrangement are easy to imagine.

To ensure that a workstation always has a current copy of a file when making changes, an application can do one of two things. On the one hand, an application can simply lock the data so that only one user has access to it at a time, whether for reading or for modifying. On the other hand, an application can perform a read-modify-write cycle every time the user makes a modification. This latter approach is usually the more desirable, since it promotes a true multi-user environment.

An application should perform a read-modify-write cycle in conjunction with data locking. The exact arrangement for synchronizing changes will depend on the nature of the data the application is dealing with. Typically, an application should present the data to the user and wait for the user to make a change. When a change is requested, the application locks the data and rereads it. The application can determine whether the change is feasible according to the current status of the data. Then the application has the choice of aborting the change or carrying through.

While data locking solves many problems, it also introduces others. One serious concern is that data locking can result in a deadlock between to workstations. Typically, if a workstation attempts to lock file that is already locked, the workstation will wait for the file's release. A deadlock can occur when two workstations attempt to lock the same two files together as a set. If each workstation succeeds in locking one of the files and then enters a waiting state, both workstations can remain there indefinitely waiting for the other to release its file.

These are the basic issues involved in synchronizing file usage on the network. Synchronization Services provide several ways of solving these problems. The following sections will look at each approach in detail.

Locking Files and File Sets

The simplest way to handle data locking is in terms of files. NetWare provides an automatic file-locking mechanism through its system of file attributes. However, Synchronization Services present a more powerful tool for locking files and coordinating their usage.

File Locking Through File Attributes

By default, NetWare places an automatic file lock on any file that is in use through NetWare's system of file attributes. NetWare's FLAG utility allows users to mark a file as shareable or non-shareable, as well as read-write and read-only. Developers can provide the same effect by using the calls found in File Services (see Chapter 4).

Using file attributes to lock a file allows only one user to access the file at a time. This is an easy way to protect data, and is adequate for many situations. For example, word processing text files are generally designed for a single user to access. By using NetWare's automatic file attribute, a word processing program can save itself the trouble of testing and assigning file locks.

Locking Files Manually

Synchronization Services contain a group of calls that allow an application to lock specific files individually or together as a set. These calls include

NWClearFile
NWClearFileSet
NWLockFileSet
NWLogFile
NWReleaseFile
NWReleaseFileSet

As might be inferred from the calls listed above, working with file locks consists of four separate tasks: logging, locking, releasing, and clearing.

A file server maintains a log table for each client, listing the file or files a workstation wants to lock. Taken together, the files in the log table are a file set. The file server will attempt to lock all of these files together. If any one of the files is already locked by another workstation, the attempt to lock the file set will fail.

An application can use the call NWLogFile to enter a file in the log table. The call includes a parameter, lockDirective, that allows the application to specify if the file should only be added to the log table or if the file server should go ahead and attempt to lock the file. Another parameter, timeoutLimit, indicates how long the file server should attempt to lock the file if the file is currently in use.

When an application has entered all the files it needs to lock in the log table, this file set can then be locked by making the call NWLockFileSet. Like NWLogFile, NWLockFileSet also has a parameter, timeoutLimit that controls the file server's efforts to lock the file set. Once the file set is locked, no other workstation can access any one of the files until it is released.

NWReleaseFile allows an application to release the lock on a particular file. The file remains in the log table and will be locked with the other files in the table the next time NWLockFileSet is called. NWReleaseFile affects only the file that is specified, all other locked files remain locked. The call NWReleaseFileSet releases all the files a workstation has locked.

Finally, a pair of calls, NWClearFile and NWClearFileSet, can be used to remove files from the log table of the requesting workstation. If a file is currently locked, these calls release the lock. NWClearFile can be used to remove individual files from the table. NWClearFileSet empties the entire log table.

Locking Records

Although file locking is an effective way to protect data, it may create quite an inconvenience for other users. This is especially true if a lot of data must be locked while only a tiny portion is being updated. Record locking allows an application to restrict data locking to individual records, structure, and variables.

These records can be locked while the remainder of a file is available to other users. NetWare allows an application to use either physical or logical record locks.

Physical Record Locks

Like a file lock, a physical record lock is associated with specific bytes of data on the file server's disk storage. Physical record locks can be used in conjunction with file locks to obtain a wide range of data locking features.

The following Synchronization Services are relevant to physical record locks:

NWClearPhysicalRecord
NWClearPhysicalRecordSet
NWLockPhysicalRecordSet
NWLogPhysicalRecord
NWReleasePhysicalRecord
NWReleasePhysicalRecordSet

The calls for manipulating physical record locks are equivalent to those for manipulating file locks and include logging, locking, releasing, and clearing. Like files, physical records are logged into a table that the file server uses to coordinate the record locking. An application can manage this process the same way it would manage file locking (see "Locking Files and File Sets" in this chapter).

An application can log a physical record using the call NWLogPhysicalRecord. This call passes a DOS handle for the file that contains the record. The record is indicated by passing its starting offset along with the record's length.

Another parameter, lockDirective, indicates whether the record should locked at the time it is logged. If the record is locked, it can be made available to other workstations for reading or it can be made available exclusively to the requesting workstation.

An application can lock all the records in the log table by calling NWLockPhysicalRecordSet. Individual records and record sets can be released and cleared using the calls NWClearPhysicalRecord, NWClearPhysicalRecordSet, NWReleasePhysicalRecord, and NWReleasePhysicalRecordSet. These calls work virtually the same as their file lock equivalents (see "Locking Files and File Sets" in this chapter).

Logical Record Locks

Many applications may find it simpler and more convenient to identify records logically rather than physically. A logical record is a name that represents network data. The data may include files or physical records. An application can assign logical records as it chooses, but it must consistently that data by its record name.

Logical records are logged, locked, released, and cleared the same as physical records. However, locking a logical record locks only the record name, not the data it refers to. When a workstation locks a logical record, other workstation will have to wait until the record is released before they can lock it.

Logical record locks serve more as coordinating devices than as security devices. Their effectiveness depends on the internal consistency of the application that creates them. A locked logical record cannot prevent another workstation from tampering with the locked data if the workstation knows the data's address.

Since files and physical records affect data directly, they invalidate logical record locks. For this reason, logical record locks should never be used in conjunction with file locks or physical record locks.
They following Synchronization Services control logical record locking:

NWClearLogicalRecord
NWClearLogicalRecordSet
NWLockLogicalRecordSet
NWLogLogicalRecord
NWReleaseLogicalRecord
NWReleaseLogicalRecordSet

Individual records are logged, locked, cleared, and released by passing a character pointer to the logical record name. This contrasts with physical records, which are referenced by a file handle and a record address. Logical record names can be up to 100 bytes in length including a null terminator.

As with physical records, an attempt to lock a logical record or record set is controlled by a time-out limit. When the limit expires, the file server aborts the attempt to lock the records. Another parameter, lockDirective, indicates whether the record should locked at the time it is logged.

Semaphores

Semaphores, like logical records, are labels that indirectly control network activity. In essence, a semaphore is an ASCII string with an associated value. A semaphore name can be up to 127 bytes in length. Its value may be from 0 through 127.

It is up to an application to define the influence of the semaphores it creates. Generally, semaphores are used to control access to a network resource. For example, a semaphore can be used to limit a resource to one user at a time or to a specified number of maximum users.

The following Synchronization Services deal with semaphores:

NWCloseSemaphore
NWExamineSemaphore
NWOpenSemaphore
NWSignalSemaphore
NWWaitOnSemaphore

An application access a semaphore's resource by opening the semaphore. When a semaphore is opened its value is automatically decremented by one. An application can open a semaphore by making the call NWOpenSemaphore. If the semaphore does not exist, NWOpenSemaphore will create it.

NWOpenSemaphore passes a semaphore name and an initial value. The initial value is assigned to the semaphore only if the semaphore does not already exist. NWOpenSemaphore returns a semaphore handle and an open count. The open count indicates how many applications have the semaphore open.

When an application is finished using a semaphore's resource, it must make the call NWSignalSemaphore to increment the semaphore's value. The application must also call NWCloseSemaphore to decrement the semaphore's open count by one. When a semaphore's open count reaches 0, the semaphore is deleted. Both of these calls pass the semaphore handle.

The call NWExamineSemaphore allows an application to find out a semaphore's value and its open count without attempting to open the semaphore. The semaphore value can be positive or negative (from -127 through 127). A positive value indicates that the application can open the semaphore and access its resource. A negative value indicates the number of processes waiting to open the semaphore. Based on this value, the application may have to wait until another application closes the semaphore before it can access the resource.

A application can use the call NWWaitOnSemaphore to wait for a semaphore to open. NWWaitOnSemaphore decrements a semaphore's value by one. If the value is greater than or equal to 0, then the application can access the semaphore's resource. If the value is a negative number, the application is placed in a queue.

Through a parameter, timeOutLimit, the NWWaitOnSemaphore call passes the amount of time the application wants to wait for the semaphore. When the time-out expires, the semaphore value is tested again. If it is positive, the application can access the resource. If it is negative, the application is removed from the queue and the semaphore value is incremented.

End of Chapter

Chapter 9
Transaction Tracking Services

NetWare Transaction Tracking Services (TTS) is a feature that ensures data integrity on files that otherwise would be corrupted when updates on the files are interrupted by such things as hardware failures or power outages. Transaction is defined as a set of one or more writes that must be completed together to maintain file and database integrity. TTS guarantees that all writes within a transaction will be completed or none will be completed. This chapter is divided into the following sections:

- Introduction
- Transaction Tracking Types
- Record Locking
- Transaction Backouts
- Applications, the User, and TTS

Note: Your version of NetWare for AViiON Systems may not support Transaction Tracking Services. Refer to the release notice accompanying your shipment for specific restrictions.

Introduction

The NetWare Transaction Tracking System allows multiple stations to have the protection of transactions while concurrently updating database files. If a workstation fails during a transaction, the database changes made by that workstation can be cancelled or backed out, without affecting other workstations.

Transaction tracking is most useful in multiuser software that employs record locking, such as database applications. It is less useful in single-user applications that deal primarily with entire files instead of records, such as word processors or spreadsheets.

The Transaction Process

The following steps describe how TTS tracks each write within a transaction:

1. An application writes new data to a file on a file server.

2. The file server stores the new data in cache memory. The target file on the file server hard disk is unchanged.

3. The file server scans the target file on the file server hard disk, finds the data to be changed (old data), and copies the old data to cache memory. The file server also records the name and directory path of the target file and the location and length of the old data (record) within the file. The target file on the file server hard disk remains unchanged.

4. The file server writes the old data in cache memory to a transaction work file on the file server hard disk. This transaction work file resides at the root level of the SYS volume on the file server. The file is flagged System and Hidden. The target file on the file server hard disk is still unchanged.

5. The file server writes the new data in cache memory to the target file on the file server hard disk. The target file is now changed.

The file server repeats these steps for each write within a transaction, and the transaction work file grows to accommodate the old data for each write. If the transaction is interrupted, the file server writes the contents of the transaction work file to the target file, thereby restoring the file to pretransaction condition. In effect, the file server backs out the transaction.

Installation and Operation of TTS

This section gives some guidelines for TTS installation and operation to ensure that your system runs smoothly.

Work Files and Work Volume

During NetWare installation, you must specify which volume on the file server will be used for transaction work files. This work volume must have plenty of disk space and if possible, be located on a different disk channel than the database files. If it is located on a different disk channel, backout data can be written simultaneously with database updates.

Transactions per DOS Task

A file server can monitor 100 to 10,000 transactions at a time. This value is configurable with SET. (The default is the maximum.) However, a file server can monitor only one transaction at a time for each workstation DOS task. If a task sends several transactions to a file server rapidly, the file server queues the transactions and services them one at a time.

Transaction Length

Transactions can be any length and contain any number of write requests. However, it is important to remember that large transactions require more space for the transaction work files. For example, a transaction in which an application truncated a 10 megabyte file to 1 megabyte, the transaction command would require over 9 megabytes of transaction work file space. It takes the file server about twice as long to process a disk write request to a transaction file as it does to process a conventional write request.

Just as exclusive locks should not be in force for long, transactions should not be active for long. Although transaction work files are reused, they cannot be reused until all transactions to which they correspond are completed. And if there are many large transactions or transactions of long duration, the transaction work files must be extended, consuming more space on the volume. It is unlikely that the transaction work volume would run out of space, but it is possible. If the work volume runs out of space, transaction tracking is automatically disabled and the following message appears at the file server console:

```
Transaction Tracking disabled because the volume is full.
```

This message is issued when the volume containing the transaction work files becomes full. Processing will continue as usual; however, transaction tracking is disabled from then on (the same as if disabled from the file server console).

Any transactions that write after transactions are disabled cannot be backed out if the station or file server fails. After freeing more space on the transaction work volume, transactions may be turned on again using the file server console command Enable Transactions.

Disk Cache Buffers

Transaction tracking uses extra disk cache buffers. Ideally, you should have at least 8K of disk cache buffers per station that uses the file server. An insufficient number of cache buffers causes thrashing.

Marking Files Transactional

Transaction Tracking occurs only on the files that have their transaction flag bit set. You can modify the bit with the NetWare FLAG utility, or applications can modify it using the File System APIs (NWSetFileAttributes). When this bit is set, files are referred to as transaction files. Work files or report files probably should not be marked transactional.

A file marked transactional cannot be deleted or renamed. To delete or rename such a file, flag it non-transactional. Usually, a database file is deleted or renamed before being compressed or used to generate a newer database. Forcing the user to flag a transaction file non-transactional makes the user more aware of transaction tracking. It also reminds the user to flag the new database file transactional. The application should handle this situation if it is TTS aware.

File Server System Files

The file server uses transaction tracking when updating the system files. This includes bindery and queue management files. Though no users are running TTS applications, the file server may require some transaction backouts to preserve system file integrity if the server goes down abnormally.

Spanning Multiple File Servers

Transactions can span multiple file servers. However, the explicit transaction calls must be issued to each server individually. Use the serverConnID parameter to specify which server gets the explicit transaction requests.

When spanning multiple servers, you must also consider backout windows. One file server may crash before committing the transaction, while another does not. A workstation can send an Explicit End Transaction to one file server, but crash before sending it to another. The application program must handle these problems.

Novell recommends that implicit transactions not span multiple file servers.

Non-Transactional Servers and TTS

A file server that does not support transaction tracking still returns successful completion codes when explicit transaction calls are made. This means that applications can be modified to use explicit transaction calls and still function if the application uses a NetWare server that is not supporting transaction tracking.

Transaction Tracking Types

There are two categories or types of transaction tracking: Explicit and Implicit. Explicit transaction tracking begins when an application makes an Explicit Begin Transaction function call and ends when an application makes an Explicit End Transaction function call or an Abort Transaction call.

Implicit transactions begin automatically when an application does its first logical or physical record lock on a transaction file. (By default, an implicit transaction begins on the first lock.) When all records are unlocked, the file server assumes that the transaction has ended.

The NetWare utility SETTTS and the set and get threshold function calls are provided to alter the number of locks required to start and end an implicit transaction. For example, some multiuser applications may always keep one or more records locked.

Implicit transaction tracking is designed to work transparently with existing multiuser software that uses record locking (physical or logical, NetWare or DOS). All the user must do is flag the multiuser database files as transactional; everything else is automatic. However, implicit transactions are not guaranteed to work with all multiuser applications. For example, applications that do not synchronize file updates exclusively with record locks and applications that synchronize updates improperly may not work.

Explicit transaction tracking has an advantage over implicit transaction tracking in that it allows applications to determine precisely when updates within the transaction are written to disk. In addition, Explicit Begin Transaction and Explicit End Transaction calls within the application allow the developer to identify the beginning and end of file update sequences.

Identifying the beginning and end of file update sequences makes it easier for applications to group file updates and practice correct record locking practices. Correct record locking practices include locking and unlocking records as a group to avoid deadlock, and not leaving records locked exclusive while waiting for keyboard input.

Modifying an application with explicit transaction calls does not affect its operation in a non-NetWare environment.

Record Locking

Record locking provides security and data protection during transactions. If a record is not locked by an application but is written to during a transaction, the file server physically locks that record automatically. This physical lock remains in force until the transaction is completed. Physically locking written records is an added protection that prevents other workstations from examining or modifying them while they are being changed.

The file server usually holds logical and physical record locks on a file until the end of the transaction, even if the file is unlocked by a workstation before the transaction is completed.

For example, if a workstation requests an unlock before a transaction is completed, it will get a SUCCESSFUL completion code indicating that the records are unlocked; however, the lock is actually held until the transaction is complete. The one exception to this is a file that is not updated while the lock is in force. A request to unlock such a file is honored.

The file server delays record unlocking because the data controlled by the locks could still be changed by a transaction backout. Thus, the file server guarantees that other workstations will not see data that is being changed until those changes are final. If multiple stations attempt to change a file, only the first station is allowed to make the change. Usually, multiuser software synchronizes and prevents other workstations from examining records that are being changed.

If a workstation attempts to read from or write to a record that is physically locked by the file server, the workstation gets a "locked" error. This means that applications which do logical record locks can potentially get unexpected physical lock errors. However, because unlocking logical records is also delayed during a file write, this should never happen if the logical record synchronization is correct. The logical record locks keep other workstations away from the physically locked records.

It is important to point how that in a non-TTS environment it is valid to unlock some updated records in the middle of a transaction if they have been completely updated. In a TTS environment, however, those records cannot be unlocked because they can still be changed--not by the application but by a transaction backout.

Transaction Backouts

Transactions are backed out because of system failures resulting from hardware problems and power outages at the workstation or the file server. But backouts also occur because of problems with applications running on a workstation or because of user intervention at a workstation.

Causes

Below is a list of some common causes for a transaction backout.

- Transaction backout occurs if an application terminates while a transaction is in progress (a begin transaction with no end transaction). For example, the user may enter a CTRL-Break.
- Transaction backout occurs if, after terminating, an application leaves records locked.
- Transaction backout occurs if there has been no activity from a station for 15 minutes. If a server does not receive packets from one of the workstations listed in its Connection Table for more than five minutes, it sends a Watch Dog packet to that station. If the station does not respond, the server continues to send a packet every minute for fifteen minutes.
- After fifteen minutes, if the workstation still does not respond, the server logs it out. Usually lack of response from a workstation indicates a power failure, software hang, or a problem with an intermediate network route, etc.

- Transaction backout occurs if a station re-attaches to the same server after rebooting and reloading the shell. If the station attaches to a different server after a reboot, the file server Watch Dog process will re-initialize the station after 5 minutes.
- Transaction backout occurs if a CLEAR STATION or a DOWN command is issued at the file server system console.

Solutions

The following are some general suggestions of how to recover from a workstation or file server that goes down and therefore backs out a transaction.

Workstation

When a workstation goes down in the middle of a transaction the user may not know exactly which transaction was completed and which was backed out. However, unless the application is multi-tasking, it should have only one transaction active at the time of the crash. Even if a workstation goes down, transactions that were completed but not yet written to disk will be written to disk and will not be backed out.

If the station goes down and transactions were backed out, the user must ensure that the last transaction was complete, or if it was not complete, do it over again. If could be partially completed if the application incorrectly used several transactions per operation. The following message will appear at the file server console if transactions were backed out:

```
Transaction being backed out for station ##.
```

This message is given at the file server console if a transaction is backed out for any reason other than an Abort Transaction from the application. Note that if the station had several tasks active, several messages are given. The file server Down command can force several of these messages to be issued.

If a workstation goes down and cannot be brought back up, the user should issue a CLEAR STATION command at the file server console. Besides affecting the transaction tracking system, the downed station keeps its records locked, which may affect other stations. If the CLEAR STATION command is not issued, the file server Watch Dog process will clear out the station after 15 minutes.

File Server

Unlike the workstation, if the file server goes down before transactions have been written to the disk, it will back out any incomplete transactions when it is rebooted.

If your file server does go down and files are backed out, the following message will be issued at the file server console:

```
### transactions need to be backed out. Type a character to start
the backout.
```

This message will appear when the file server is being brought up (after all of the volumes have been mounted). The transaction tracking software has detected that some of the transactions were not completed before the server went down. You should not see this message unless the server went down abnormally.

After bringing up a file server that crashed, the user is responsible for determining which updates were completed and which were backed out and need to be repeated. A good practice is to verify that the last few changes were actually made. Remember that most transactions should be committed to disk within 3 to 5 seconds of being completed. However, any operation completed within 10 seconds of a failure should be examined. After the file server is brought up, a good practice is to have the workstation operators review their last operation. If it is not completed, they should also review the next to last operation, and so on.

If the NWTTSIsTransactionWritten call is used by the application, the user will have a better idea of when transactions were actually completed. However, just because the application thought that a transaction was not completed does not mean that it wasn't. The transaction could have been completed between the last time the application checked and the time the file server crashed.

Disable/Enable Transaction Tracking

Transaction tracking can be disabled or enabled from the file server console with the commands Disable Transactions and Enable Transactions. However, even when transaction tracking is disabled, tracking occurs in a partial way:

- The Explicit Begin and End Transaction calls still denote transactions and work correctly.
- The NWTTSIsTransactionWritten call still returns correct status.

If TTS is disabled, the only loss to explicit and implicit transaction calls is the capability of backing out incomplete transactions. For example, if a file server or workstation fails while transaction tracking is disabled, you cannot back out because the backout information will not have been saved, and the database could be corrupted.

When transaction tracking is re-enabled at the system console, backout capability will only be provided to new transactions. Transactions that are in progress when transaction tracking is re-enabled won't have backout, and the possibility of file corruption still exists until these unprotected transactions are completed and written to disk. It is important to note that a transaction can be backed out if the workstation does not write to a transaction file after transactions are disabled. If a workstation does a write after transactions are disabled, the transaction is invalidated and any previous backout information for that transaction is ignored.

Disable Transactions

DISABLE TRANSACTIONS (NetWare 2.x), DISABLE TTS (NetWare 3.x) and NWTTSDisableTransactionTracking (NetWare for AViiON Systems) are commands that turn off the backout capability of the transaction tracking system. Any transactions written after transactions have been disabled cannot be backed out. A transaction that has not written anything since transactions were disabled can still be backed out while transactions are disabled. Normally, this command is used only for testing.

Enable Transactions

ENABLE TRANSACTIONS (NetWare 2.x), ENABLE TTS (NetWare 3.x), and NWTTSEnableTransactionTracking (NetWare for AViiON Systems) are commands that re-enable transaction tracking. Any previous transaction backout information is erased, except those that were active at the time transactions were disabled and did not write anything. These transactions are not erased and can still be backed out.

Applications and TTS

Almost all applications should work correctly with TTS implicit or explicit transactions, or be easily adapted to do so. However, the following TTS record locking features discussed earlier could adversely affect certain applications:

- Physical and logical record locks remain in force until the end of the transaction.
- Records unlocked in the middle of a transaction are usually held until the transaction completes.
- The file server automatically generates a physical lock when a record is written if the record is not yet locked.

Explicit Transactions: Deadlocking

Consider, for example, the following application running without TTS: The application locks a "header" record. It then needs to allocate data records in several files. It does this by locking an "avail" record in one of the data files, allocates the record, changes the record, and then unlocks the "avail" record.

It continues in this manner, allocating another data record in another file, never locking more than one "avail" record at a time. Having only one "avail" record locked at a time is the way the application avoids deadlocks.

However, when the same application is run with TTS enabled, the file server keeps the "avail" records locked (even after an unlock request) because the "header" record is still locked and the transaction is still going. Meanwhile, the application thinks it has unlocked them.

Thus, one workstation locks "avail" record A and another locks "avail" record B. When the stations request an unlock on their "avail" records, they will get back SUCCESSFUL completion codes even though the records remain locked. Then, when the first station attempts to lock "avail" record B, and the second station attempts to lock "avail" record A, the workstations will deadlock. An application that runs fine without transaction tracking cannot run with it.

The same types of deadlocking problems can occur if multiple stations attempt to extend the same transaction file. This is because the file server delays the completion of the second file extend until the first transaction is completed.

There are several ways applications can be changed to overcome these problems with transaction tracking.

- Deadlock can be avoided if the application always locks the "avail" records in a certain order ("avail" A, if ever locked, is ALWAYS locked before "avail" B).
- Deadlock is also avoided if the NetWare "lock record set" command is used to lock all "avail" records at once.

Most multiuser application will not have deadlocking problems with TTS, because the cases that cause deadlock are obscure.

Implicit Transactions

Implicit transactions should work correctly with almost all multiuser software that uses record locks, except in the following circumstances:

Semaphore Synchronization

Multiuser packages that synchronize using methods other than record locks may not work with implicit transactions. For example, if an application uses semaphores or data in a file to synchronize, it may not correctly signal implicit transaction begins and ends to the file server.

Single or Multiple Transactions

The user and the application may view what are actually several transactions as a single transaction, while TTS treats the transactions separately. For example, suppose that to add a new customer, the user first adds the customer to the "billings" database files (one transaction), then adds the customer to the "order entry" database files (another, separate transaction). The user assumes there will be no problem adding to the "billings" file and also to the "order entry" file.

However, if the file server (or workstation) goes down with the customer added to the "billings" files and not to the "order entry" files, the user or the application may not detect this inconsistency; the database would remain inconsistent. Even if the problem is detected, the application may not allow the user to add the customer only to the "order entry" files, or delete the customer only from the "billings" file.

Premature Unlocking

Transactions should not continue after all locks are released because once all records in a transaction are unlocked, the application has lost control over the data. For example, an application may want to lock additional records to finish a transaction, but be unable to get the records locked. If this is the case, it should back out the changes it has already made. Even if an application could get additional records locked to complete a transaction, those records may have already been changed by another workstation.

Some applications may know that after they unlock all of their records, the rest of the transaction will always work and never have lock problems. However, these types of applications will only work with explicit transactions.

Transactions Too Large

If an application package never releases all of its record locks, everything is done as one large implicit transaction. It still works fine, but the granularity of the different updates is lost. Large transactions tie up disk space in the transaction work files. Having too many large transactions active could possibly fill up the volume containing the transaction work files. If this occurs, transaction backouts have to be disabled. Note that this type of package can also overflow the record locking tables, because most of the physical and logical record locks are not released until the transaction is completed.

If an application needs to keep a lock (for example, to control the number of users the application supports), the solution may be to not flag the file with permanent record locks as transactional.

Another possible solution is to add explicit transaction begin and end requests. And a third solution is to use the Set Implicit Transaction Record Lock Threshold function and set the record lock threshold to the number of records permanently kept active by the application.

Applications, the User, and TTS

Novell is encouraging multiuser software developers to test, verify and certify that their software is TTS compatible. These developers may wish to publish a set of special instructions for running their package with TTS. TTS-aware applications can often minimize the user's need to understand TTS, but there are some things, particularly file server failure recovery, that require the user to understand how TTS works even with a TTS aware application.

The user also needs to understand how to handle transaction files. In particular, for example, these files cannot be deleted or renamed as explained in the Introduction. Also, if the file is moved or restored from backup, the transaction attribute will be lost and will need to be restored.

It is also important for users to know that certain operations can be performed more efficiently without TTS. A good example is data file compression. Some databases require their data files to be compressed periodically to recover deleted records. The compression is performed by only one workstation and results in a new database being generated.

Before the compression, the user should flag the database files non-transactional so that all of the compression operations do not become one gigantic transaction. (Some database software uses file locks instead of record locks during compression, and will not have this problem.)

It is important for the user to understand the nature of compression and the fact that a new database file is generated. If application software were not TTS aware, the new database file would not be flagged by the application as transactional. The user must remember to handle this case.

The user needs to be aware of the TTS work file volume and its memory requirements.

The user needs to determine which files need to be flagged transactional. This implies the user knows the purpose of each application file. All shared database files including index files should be marked transactional. Failure to mark all files violates a fundamental assumption of the TTS system and will cause transaction backouts to be incomplete.

The user needs to know how to operate the FLAG utility.

End of Chapter

TIPS ORDERING PROCEDURES

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a) MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

      Send your order form with payment to:
      Data General Corporation
      ATTN: Educational Services/TIPS G155
      4400 Computer Drive
      Westboro, MA 01581–9973

   b) TELEPHONE – Call TIPS at (508) 870–1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a) Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.
   b) Check or Money Order – Make payable to Data General Corporation.
   c) Credit Card – A minimum order of $20 is required for Mastercard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	Shipping & Handling Charge
1–4 Units	$5.00
5–10 Units	$8.00
11–40 Units	$10.00
41–200 Units	$30.00
Over 200 Units	$100.00

If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	Discount
$1–$149.99	0%
$150–$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

6. Allow at least two weeks for delivery.

RETURNS

7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870–1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	SHIP TO: (No P.O. Boxes - Complete Only If Different Address)
COMPANY NAME _____	COMPANY NAME _____
ATTN: _____	ATTN: _____
ADDRESS _____	ADDRESS (NO PO BOXES) _____
CITY _____	CITY _____
STATE _____ ZIP _____	STATE _____ ZIP _____

Priority Code _____ (See label on back of catalog)

_____ _____ _____ _____ _____
Authorized Signature of Buyer (Agrees to terms & conditions on reverse side) | Title | Date | Phone (Area Code) | Ext.

ORDER #	QTY	DESCRIPTION	UNIT PRICE	TOTAL PRICE

A SHIPPING & HANDLING

☐ UPS	ADD
1-4 Items	$ 5.00
5-10 Items	$ 8.00
11-40 Items	$ 10.00
41-200 Items	$ 30.00
200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0 - $149.99	0%
$150 - $499.99	10%
Over $500.00	20%

Tax Exempt # or Sales Tax (if applicable) _____

ORDER TOTAL	
Less Discount See B	-
SUB TOTAL	
Your local* sales tax	+
Shipping and handling - See A	+
TOTAL - See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is _____. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Expiration Date ☐☐☐☐

_____
Authorized Signature
(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508-870-1600.

Form 702
Rev. 8/87

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. CUSTOMER CERTIFICATION

Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

2. TAXES

Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. DATA AND PROPRIETARY RIGHTS

Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

4. LIMITED MEDIA WARRANTY

DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY

EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY

A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. GENERAL

A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts, excluding its conflict of law rules. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)

Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision-locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

NetWare® for AViiON® Systems: C Interface Programmer's Guide

069–000566–00

Cut here and insert in binder spine pocket