File Services

File Services enable applications to manipulate file system information. Because the NetWare C Interface for Univel 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; 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 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 ªPathsº on the next 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 delete 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 the NetWare C Interface for Univel
Reference Manual. 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 bit set. 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.

The following table gives examples of wildcard strings and file names that would and would not match them.

TABLE 21. Wildcard Patterns

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

		AB ABC	
^.	AB^.	AB AB.	ABC AB.C

Entry IDs

Entry IDs are used with many of the file system API calls. They signal the file server to search for a 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 the search attributes parameter. These functions always act on normal files, whether the hidden or system bit is set or not. The search attributes 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 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 illustrated in the following table.

TABLE 22. Security Access 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 logged in to the server as supervisor or as an object with supervisor security equivalence. Both a read and a write bit can be set.
NetWare	Access allowed only to NetWare.

For more information, see Bindery Services.

Effective Rights

Effective rights are the rights a user can actually exercise in a given directory or file. Effective rights are determined differently for NetWare v2.*x* and NetWare v3.*x*. (For NetWare v2.*x*, see Appendix A.)

To determine a user's effective rights in NetWare v3.*x*, 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. 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 right in a directory can grant other clients Supervisory rights to the directory, its files and its subdirectories. Once the Supervisory right 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 subdirectory'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 ªinheritº 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, and scan and set file trustee assignments. Each task is described below except scan and set trustee assignments.

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

The NWMoveEntry 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 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 existing file.

The NWCloseFile 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's attributes. Two functions, NWScanFileEntryInfo and NWSetFileEntryInfo, are discussed in the next section. These two functions allow you to scan and set file attributes as well as many other file information fields.

The NWGetFileAttributes 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 the specified file, and the file must not be in use by other clients. This call supports wildcard attribute setting.

The Execute Only attribute cannot be reset. Setting this attribute requires the client to have supervisor equivalence.

Scanning and Setting File Information

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

The NWScanFileEntryInfo 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 allow you to scan or change only that item.

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

Since the functions listed 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 ªIDs and Usersº and ªDate and Time Informationº sections.) Other items require the following rights:

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 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.

F0.eps ,

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.

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 only valid for NetWare version 3.0 or above. 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.

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 wildcard 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 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 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, creation, archive, and 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 ªtº 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 (see below for usage).

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

This call is only valid for NetWare version v3.*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 only available for NetWare version 3.0 or above. 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 only valid for NetWare v3.*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 two parameters as shown below.

```
NWSetDirsInheritedRightsMask( &path, newRightsMask )
```

This function will revoke directory rights by removing from the directory's

Inherited Rights Mask any rights that the application does not pass in with the newRightsMask parameter. It will grant rights by adding to the directory's
Inherited Rights Mask any rights passed in with the newRightsMask parameter.

Volume Manipulation

A NetWare v3.*x* file server can have up to 32 disk volumes installed. 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 v3.*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 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 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 4K blocks. This call is only valid on NetWare v3.*x*. The volume number can be obtained by calling NWGetVolNum.

Getting Volume Numbers

The NWGetVolNum 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 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 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 AND'ing the Inherited Rights Mask of the directory with the client's current trustee rights.

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 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 OR'ing 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.

Scanning Salvageable Files

The NWScanSalvageableFiles returns information on deleted but salvageable files. This call is only valid for NetWare version v3.*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 v3.*x*.

This call 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 in only valid on NetWare v3.*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.