MULTIPROTOCOL DATA ACCESS: NFS, CIFS, AND HTTP

TR3014

by Andy Watson, Paul Benn, and Alan G. Yoder,

updated by H.T. Sun, Network Appliance, Inc.

TECHNICAL REPORT

Table of Contents

Table of Figures

1. Introduction

Network Appliance™ filers (file server appliances) provide fast, simple, and reliable network data access to Network File System (NFS), Common Internet File System (CIFS, for Microsoft® Windows® networking), and Hypertext Transfer Protocol (HTTP, primarily for Web browsers) clients. Support for all three protocols is woven into the NetApp microkernel and file system, providing multiprotocol data access that transcends the enclosed perspective of general-purpose operating systems.

In the context of file service to Windows clients, Data ONTAP™ software for Windows is virtually indistinguishable from other Microsoft Windows servers in a Windows domain. For example, in addition to many other Windows-compatible features,

- Access control lists (ACLs) can be set on shares, files, and directories
- NetApp filers can be administered via Windows Server Manager and User Manager
- UNIX® users are mapped to Windows users
- Multilanguage support is available via UNICODE
- File access logging can be tracked for Windows and UNIX users
- Filers interoperate with NTFS and Active Directory.

Windows-style ACLs and UNIX-style file access permissions are fully integrated on the NetApp filer. Furthermore, Windows users are automatically mapped on the fly to their respective UNIX accounts (to assess file permissions), simplifying the unification of the two separate namespaces. This is especially powerful in conjunction with the NetApp *autohome* feature, which provides all Windows users with share-level access to their own home directories without the painstaking administrative efforts typically required on other Windows file servers. (Each user automatically sees his or her own home directory as a share in the Network Neighborhood—but not other users' home directories, unless those others have been deliberately and explicitly exported as publicly visible shares, of course.)

With multiprotocol filing, PCs[1] can store and access data side by side with UNIX-based clients, without compromising their respective file attributes, security models, or performance. Users with PC desktops can work within the single instances of their home or project directories, with Windows-based applications executing locally, or UNIX-based applications running on a server. And whether written to the filer via NFS or CIFS, documents can be accessed directly by a wide variety of Web browsers via HTTP.

[1] Throughout this document, the term "PC" implies "Microsoft Windows networking client" unless indicated otherwise.

Multiprotocol filing liberates the *data infrastructure*, largely freeing it from the constraints of operating system preference or legacy investments. This paper

- describes the NetApp multiprotocol filer architecture
- explores the implications of multiprotocol filing for system administrators and end users
- reflects the Data ONTAP software for Windows evolution

2. File System Permissions

NetApp filers support both UNIX-style and NTFS-style file permissions. Because the ACL security model in NTFS is richer than the file security model used in UNIX, no one-to-one mapping can be made between them. The fundamental problem occurs when a PC client—which expects an ACL—accesses a UNIX file, or a UNIX client—which expects UNIX file permissions—accesses a PC file. In these cases the file server must sometimes authorize the request using a user identity that has been mapped from one system to the other, or in some cases even a set of permissions that has been synthesized for one system based on the actual permissions for the file in the other system. The NetApp filer's promise to its users is that these synthesized file permissions are at least as restrictive as the true file permissions. In other words, if user `agy` cannot access a file using the true file permissions, the same is true when using the synthesized file permissions. Data ONTAP has a mechanism called *UID-to-SID mapping* to address this issue (see Section 3.6).

2.1 UNIX File Permissions (UFS)

UNIX file permissions are usually represented as three sets of concatenated rwx triplets. An example directory listing in a UNIX file system looks like this:

```
lrwxrwxrwx 1 agy eng 10   Sep 2  14:42   perms.doc->perms.html
-rw-r--r-- 1 agy eng 1662 Sep 2  14:32   perms.html
-rw-rw---- 1 agy eng 2399 Feb 19 1998    privileges.nt.txt
drwxr-xr-x 2 agy eng 4096 Sep 2  14:42   work
```

The first 10 characters on each line indicate the file type and permissions for the listed file. The first character contains a `d` if the file is a directory, and an `l` if it is a symbolic link, and a dash (**-**) if it is a regular file. The next three characters specify whether the user (`agy` in this example) can read(`r`), write(`w`), or execute(`x`) the file. The following three characters specify the permissions for the group associated with the file (`eng` in this example). The last three characters specify the permissions for users who are neither the owner nor members of the file's group. In the example, `perms.doc` is a symbolic link anyone can traverse (obtaining the file `perms.html`), `perms.html` is a regular file anyone can read but only the user `agy` can write, `privileges.nt.txt` is a file `agy` or anyone from the group `eng` can both read and write, and `work` is a directory that anyone can search and read files in, but only `agy` can insert files into or delete files from.

When user `agy` attempts to access a UNIX file named `nfsfile`, the behavior of the file system depends on what kind of file it is. First, though, the request is checked against the permissions associated with the file. Suppose it is a read request. If `agy` is `nfsfile's` owner and the owner has read permission on `nfsfile`, the request can be honored. Otherwise, if `agy` is a member of the file's group and the group has read permission, the request can be honored. Otherwise, if all others have read permission, the request can be honored. If none of the foregoing tests succeed, the request is denied.

2.2 PC File Permissions (NTFS)

PCs use a different system for denoting file permissions. On the FAT and FAT32 file systems—which were designed as single-user file systems—there are no permissions; anyone who can gain access to the machine has unlimited privilege on every file in the system. The NTFS file system, however, available from Microsoft only on workstation and server platforms running Windows NT® and Windows XP, 2000, and 2003, has a sophisticated security model. This same security model is also available for use by the CIFS network file system protocol on NetApp filers, so PC clients accessing files on a filer, whether or not they are running NTFS can also use this security model. For more detailed discussion of CIFS, see Section 3 of this document.

In NTFS and CIFS, each file has a data structure associated with it called a *security descriptor* (SD). This contains, among other things, the file owner's *security ID* (SID) and another data structure called an *access control list* (ACL, usually pronounced "ackle"). An ACL consists of one or more *access control entries* (ACEs), each of which explicitly allows or denies access to a single user or group.

Suppose user `agy` attempts to open file pcfile for reading. The algorithm used to determine whether to grant `agy` permission to do this is conceptualized as follows:

- First search all the ACEs that deny access to anyone. If any of them deny read access to `agy` specifically or to any of the groups of which `agy` is a member, stop searching the ACL and reject the request.
- If no denials of access are found, continue searching the rest of the ACEs in the ACL. If one is found that grants read access to `agy` or to any of the groups `agy` is in, stop searching the ACL and allow the request.

- If the entire ACL has been searched and no ACEs were found that allow `agy` to read the file specified in the request, reject the request.

It should be clear by now why it is not always possible to make a one-to-one mapping from the ACL model to the UNIX security model. For example, it is possible using the ACL security model to allow access to all the members of a group except some specified user. This can't be done using the UNIX model.

2.2.1 NTFS Access Modes

The Windows file permissions model defines more access modes than UNIX does (read, write, and execute). The following table explains what each of the basic file access modes means.

Request Type	The File Is a Folder	The File Is a Normal File
Read (r)	Display the file's data, attributes, owner, and permissions	Display the file's data, attributes, owner, and permissions
Write (w)	Write the file, append the file, and read or change its attributes	Write the file, append the file, and read or change its attributes
Read & execute (x)	Display the folder's contents; display the data, attributes, owner, and permissions for files within the folder; and run files within the folder	Display the file's data, attributes, owner, and permissions, and run the file
Modify	Read, write, modify, and execute files in the folder; change attributes and permissions; and take ownership of the folder or files within	Read, write, modify, execute, and change the file's attributes
Full control	Read, write, modify, and	Read, write, modify,

	execute files in the folder; change attributes and permissions; and take ownership of the folder of files within	execute, and change the file's attributes and permissions and take ownership of the file
List folder contents	Display the folder's contents; display the data, attributes, owner, and permissions for files within the folder; and run files within the folder	

Windows XP, 2000, and 2003 also support special access permissions, which are made by combining the permissions described above. The following table shows these special access permissions and their combinations.

File Special Permissions	**Full Control**	**Modify**	**Read & Execute**	**Read**	**Write**
Traverse folder/execute file	X	X	X		
List folder/read data	X	X	X	X	
Read attributes	X	X	X	X	
Read extended attributes	X	X	X	X	
Create files/write data	X	X			X
Create folders/append data	X	X			X
Write attributes	X	X			X
Write extended attributes	X	X			X
Delete subfolders and files	X				

Delete	X	X			
Read permissions	X	X	X	X	X
Change permissions	X				
Take ownership	X				
Synchronize	X	X	X	X	X

3. File Service for Heterogeneous Environments with NFS and CIFS

NetApp filers support both NFS-style and CIFS-style file permissions. NFS-style file permissions are widely used in most UNIX systems, while CIFS-style file permissions are used in Windows when communicating over networks. Because the ACL security model in CIFS is richer than the NFS file security model used in UNIX, no one-to-one mapping can be made between them. This mathematical fact has forced all vendors of multiprotocol file storage products to develop nonmathematical strategies to blend the two systems and make them as compatible as possible. This section explains the NetApp approach to this problem.

File service for heterogeneous environments (UNIX workstations plus PCs) is challenging. PC NFS software can be installed on PC clients or SAMBA can be installed on UNIX server, but these approaches are either costly or time-consuming or introduce an extra layer of file system emulation. Common sense suggests that changing the file server makes better sense than altering a large (and growing) number of PC clients or adding a file system emulation layer that reduces performance. In other words, in a heterogeneous environment, the file server should support remote file access protocols for both UNIX-based clients and PCs.

The alternative—using separate file servers for each protocol—can increase costs due to administrative complexity and redundant investments in storage. Routine administrative functions like backup and restore are duplicated. And it is still difficult to implement applications that need to facilitate sharing of data between UNIX and PC users.

Perhaps worst of all, perpetuating an arrangement of separate servers for distinct sets of UNIX and PC clients creates an awkward situation for users who need to access the same files (in their home directories, for example) with locally executing applications on their PC, *and* by means of an X-Windows session on a UNIX host (see Figure 1).

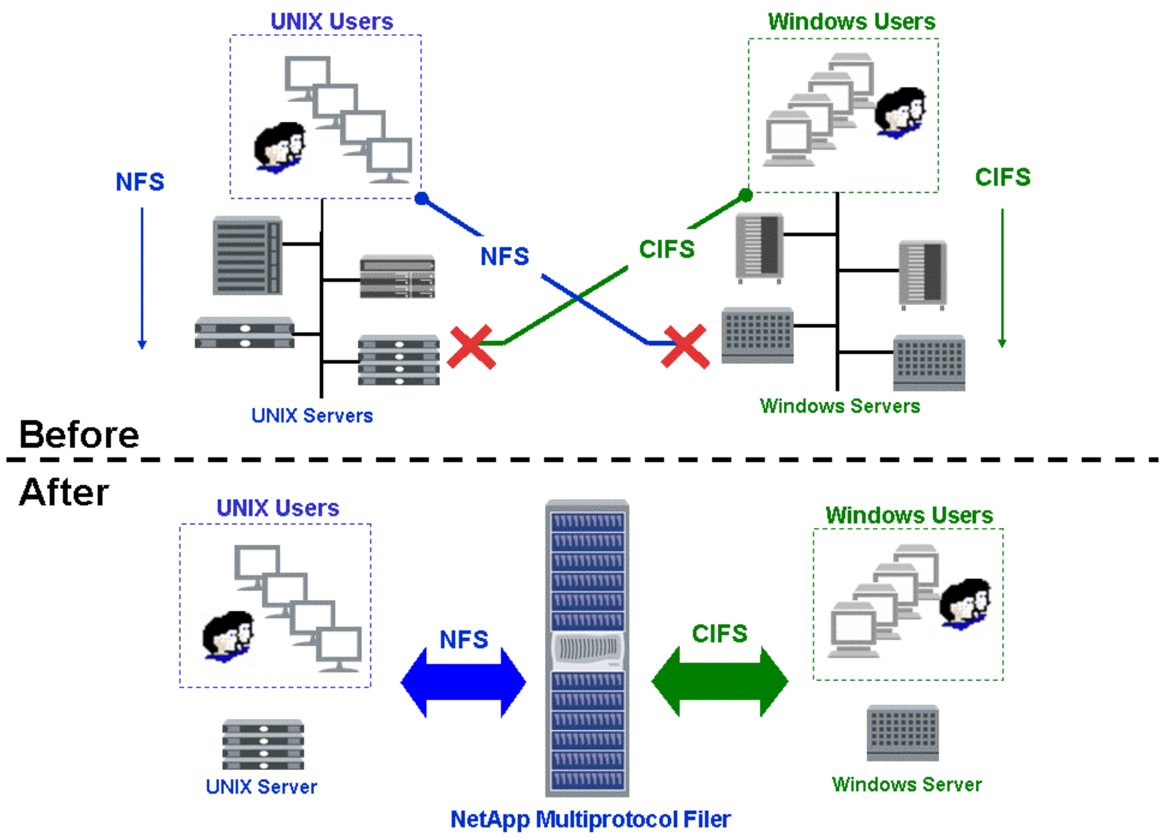


Figure 1) Multiprotocol NetApp filer.

3.1 NFS

Most UNIX clients use NFS for remote file access. Sun Microsystems introduced NFS in 1985. Since then, it has become a de facto standard protocol, used by 10 million systems worldwide. NFS is particularly common on UNIX-based systems, but NFS implementations are available for virtually every modern computing platform in current use, from desktops to supercomputers. Only when used by UNIX-based systems, however, does NFS closely resemble the behavior of a client's local file system. NFSv4 addresses some weaknesses in earlier NFS versions such as ACL, security, and file system namespace, etc. However, this is beyond the scope of this document, our discussion will focus on NFSv2 and v3. For more detailed information regarding NFSv4, refer to TR3085.

3.2 CIFS (SMB)

The operating systems running on PCs do not include NFS. Instead, the protocol for remote file access is CIFS, formerly known as Server Message Block (SMB). SMB was first introduced by Microsoft and Intel in the early 1980s, and is the protocol used in several diverse PC network environments. In 1992, SMB was ratified as an X/Open specification [XO- 92a and XO-92b][2].

In mid-1996, Microsoft began to promote CIFS as an open standard, with a published specification [MS-96c][3]. Through developers' conferences and interaction with standards bodies, Microsoft actively solicited input for the future evolution of CIFS, with one goal being support for the protocol on non-Windows operating systems.

3.3 NFS vs. CIFS

In every textbook description of NFS, its *statelessness* is emphasized. NFS operations are idempotent (can be repeatedly applied harmlessly), or if non-idempotent (file deletion, for example) are managed safely by the server. Clients are oblivious to server restarts (if service is restored promptly), with few exceptions. The NFS protocol emphasizes error recovery over file locking—error recovery is simple if no state need be preserved.

A CIFS file server is *stateful* (not stateless). The CIFS protocol emphasizes locking over error recovery, because PC applications rely on strict locking. Strict locking requires a sustained connection. It is imperative that an active session not be interrupted. Applications executing on PC clients react to a CIFS server in exactly the same manner as they do to local disk drives: a down server is no different from an unresponsive disk drive. Therefore, PC clients must be warned—and allowed time—to gracefully disengage (i.e., save files, exit applications, and so on) before server shutdowns or restarts.

3.4 Mixing NFS and CIFS

Software solutions exist which allow UNIX-based servers to provide remote file access functionality to PCs without requiring (PC) NFS. Running in user-mode (not in the UNIX kernel), these applications support PC clients via CIFS. Of these, the most widely used are SAMBA, Hummingbird NFS Maestro, and Windows Service for UNIX (SFU) by Microsoft. SAMBA is a server-side installation, while NFS Maestro and SFU are NFS emulators installed on the clients running NTFS.

The most prevalent application is SAMBA. For users with a casual need for CIFS access, or who are new to PCs and are trying to get a feel for what PC service is like, SAMBA offers several advantages: it is free, easily available, runs on most popular UNIX systems, and is relatively reliable for simple uses. To meet more serious requirements (e.g., providing primary file service for a large organization), SAMBA falls short in several important areas: shallow integration with the underlying UNIX-centric file system (particularly with respect to locking mechanisms), difficulty of installation, configuration, and administration, and lack of reliable support (it being public domain software).

[2] [XO-92a] "Protocols for X/Open PC Internetworking: SMB, Version 2," X/Open, ISBN 1-872630-45-6, 1992.
[XO-92b] "IPC Mechanisms for SMB," X/Open, ISBN 1-872630-28-6.

[3] [MS-96c] Microsoft's CIFS protocol specification reference.

For discussion purposes, this paper uses the more widely implemented SAMBA as the example for *emulated* CIFS protocol support, in contrast to the Network Appliance *native* multiprotocol approach.

3.5 Multiprotocol File Service

A file server cannot easily deliver functionality to clients beyond what its local file system provides. For example, the UNIX File System (UFS) does not store the "creation" timestamp for a file. A PC client cannot retrieve that information via the CIFS protocol if the server doesn't have that information to provide.

Unsurprisingly, the NFS protocol doesn't offer mechanisms for data access beyond the capabilities of UFS. NFS was developed to extend UFS across networks. Similarly, the CIFS protocol extends a PC-oriented file system to remote clients. In both of these cases, the remote file access protocol implementation is described as "native" to the operating system context in which it originated.

3.5.1 Emulated Multiprotocol File Service

When application software like SAMBA provides remote data access to the files on a UNIX file server via the CIFS protocol, it must do more than simply provide semantically correct responses in its communication with clients over the wire. It must make up the difference between the PC's requirements and the intrinsic facilities of the server's local file system (UFS).

On a file-by-file basis it must store additional information—in supplementary files if the file system itself has no provision for it. For example, the server must offer case-insensitive file name lookup for PC clients. For older PC clients, the server must also generate DOS-style "8.3" file names (up to eight characters, plus up to three characters for an optional suffix). An "8.3" file name is not inherently included in UFS, so the CIFS emulation application must store it elsewhere. Similarly, UFS does not provide case-insensitive file name lookup; the CIFS server emulation application must do this for itself.

The mapping from the PC clients' expectations to the server's UNIX context is awkward and incomplete. The reverse—a PC serving UNIX clients via NFS application software—is similarly mismatched. This dissonance is characteristic of "emulated" remote file access protocol implementations.

3.5.2 Native Multiprotocol File Service

NetApp filers are not UNIX-based, nor are they Windows-based. The microkernel operating system and WAFL® (Write Anywhere File Layout [TR-3002]) file system were designed specifically for extensible file service. Therefore NFS, CIFS, HTTP—and in the future, additional protocols—can be implemented *natively*. There is no functionality mismatch, as with the emulated approaches, and kernel-based security and file locking enforcement are inherently stronger than user-space application software methods (Figure 2).

Note that in Figure 2, all NetApp processing of a client's request is executed within the kernel as a series of function calls, eliminating the data copies and overhead of the interprocess communication (IPC) between the separate processes in the emulated approach.

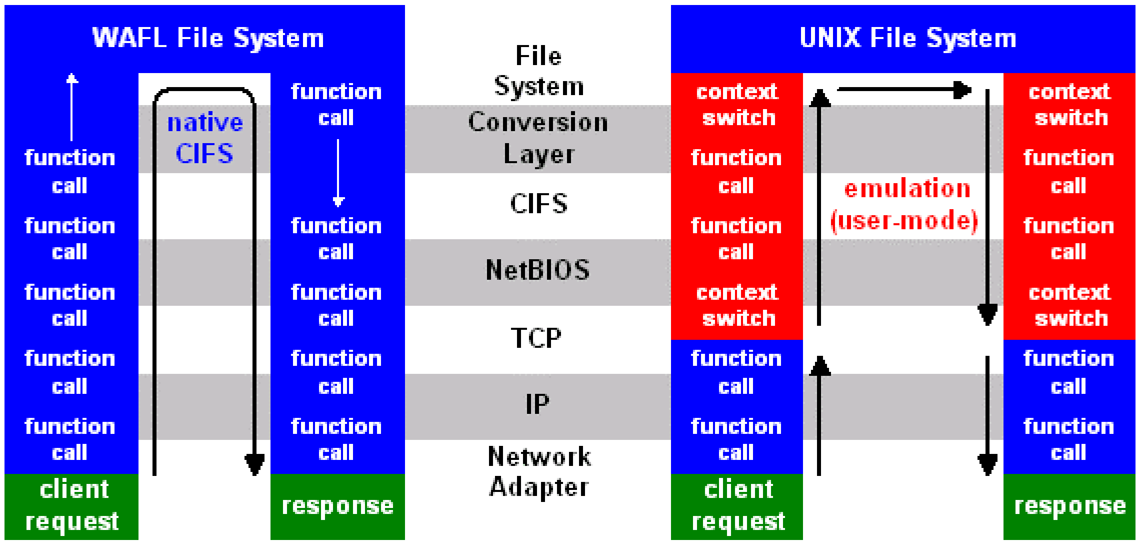


Figure 2) Processing of client requests.

The three primary elements in the Data ONTAP microkernel are: a real-time mechanism for process execution; the WAFL file system; and the RAID manager. Of these, only the RAID manager is insensitive to protocol—blocks are read and written in the same way, whether originally triggered by NFS or CIFS (or HTTP, as discussed in Section 5). Figure 3 provides a block diagram view of Data ONTAP software. Note that NFS can operate with either TCP or UDP transport mechanisms, but NetApp support for the CIFS file-sharing protocol uses TCP exclusively[4].

[4] CIFS uses NBT (NetBIOS over TCP/IP) in the NetApp implementation but can also use NETBEUI or IPX/SPX in other environments.

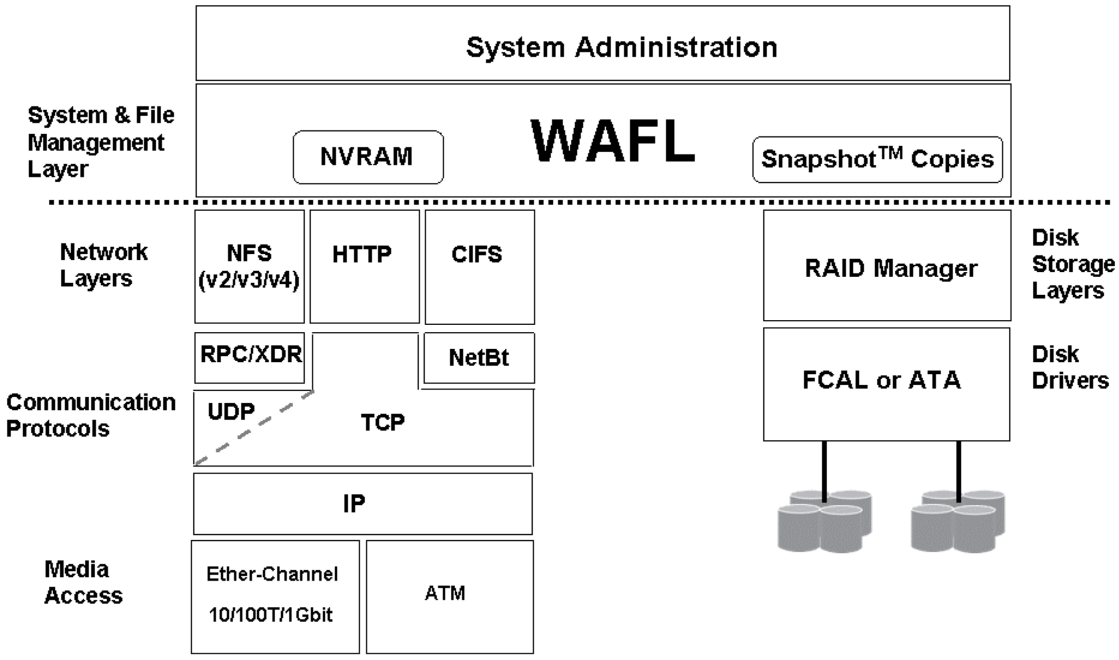


Figure 3) Data ONTAP software.

Within the microkernel, all incoming NFS, CIFS, and HTTP requests are received by the network interface driver. Once initiated, the processing of a request is uninterrupted and continuous, as far as possible, utilizing a series of function calls. This is entirely different from traditional file servers, which employ separate processes for the network protocol stack, the remote file system semantics, the local file system, and the disk subsystem. The advantages to this approach are performance and simplicity, and the simplicity enables extraordinary reliability.

The process (which begins with the network interface driver) executes continuously and blocks only when waiting is unavoidable. Ordinarily this would occur only when the request has reached the WAFL file system. By this stage, the request has been interpreted with respect to the applicable protocol—NFS, CIFS, or HTTP—and tested for correctness and legality.

If the request cannot be serviced (e.g., because it is illegal or otherwise incorrect), an error code is immediately returned. Assuming that the request can and should be serviced, a reply to the client is generated, unless one of the following conditions causes the process to block:

- A request requires data not already in memory
- An administrative event preempts other processing

Figure 4 illustrates the process flow for all cases except administrative events. (Administrative commands take effect immediately, possibly causing other operations to block.)

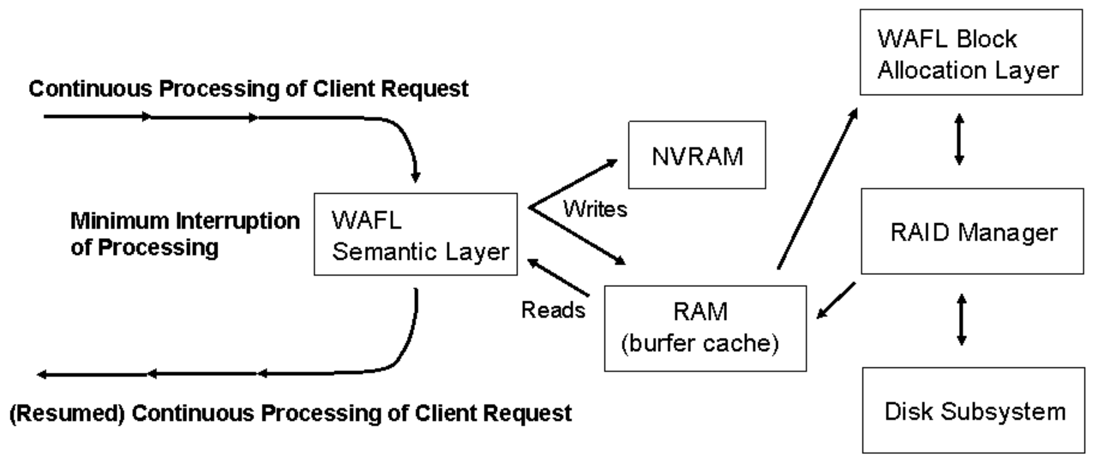

Continuous Processing of Client Request
Minimum Interruption of Processing
WAFL Semantic Layer
Writes
Reads
NVRAM
RAM (burfer cache)
WAFL Block Allocation Layer
RAID Manager
Disk Subsystem
(Resumed) Continuous Processing of Client Request

Figure 4) Process flow of handling client requests.

If a read request cannot be satisfied with data already in memory (from a previous read request, or because of read-ahead), the process will block, while WAFL requests the RAID manager to retrieve the requested data (as well as the next several blocks deeper in the file[5].

In summary, the NetApp filer serves all incoming requests with a continuous series of function calls, not IPCs, that block only when necessary, as described above. This architecture is exceptionally well-suited to a multiprotocol context because it simply

[5] Unless the `minra` option has been enabled, which limits read-ahead to a single block.

doesn't matter which protocol is used. All requests are handled expeditiously as single processes running in the kernel, regardless of protocol. Circumstances where a process might block are similar across protocols. And the WAFL file system is extensible, allowing for native accommodation of both UNIX and PC-style file and directory attributes.

3.6 How User Mapping Works

Anytime a user requests access to a file, the filer must compare the user's security information against the permissions associated with the file. The filer maps the ID from the user's native system to the ID type expected by the file's security style. UNIX requesters are authorized using UNIX permissions if the file has a UNIX security style; otherwise they are mapped the user's Windows domain account and the resulting NTFS security descriptor (SD) is authorized against the file's ACL. PC requesters are authorized using the file's ACL, if it has one; if it does not, they are mapped to a UNIX account and authorized using the UNIX permissions on the file. This is discussed in more detail later.

When performing the mapping from one security system to the other, some users and groups need special treatment. Users such as root in UNIX and Administrator in Windows servers may or may not be equivalent to each other, depending on how a site is configured. The */etc/usermap.cfg* file on the filer may be used to control the mapping between special users and groups. The usage of *usermap.cfg* file is described in detail in Section 6.3.

4. The NetApp Mixed Security Model

File security on a NetApp filer is handled by the WAFL file system, which understands both the UNIX and NTFS permissions models. The security mechanism used depends on whether the file has an ACL. There are four possible access modes:

- A UNIX client accesses a UNIX file, meaning a file without an ACL[6]
- A PC client accesses a file with an ACL
- A UNIX client accesses a file with an ACL
- A PC client accesses a file without an ACL

[6] For the purposes of this paper, all files in UNIX qtrees are assumed not to have ACLs. This is not always true, as ACLs are preserved when changing qtree styles. But any ACLs on files in a UNIX qtree are ignored when doing permissions checks, so the net effect is as if no file in a UNIX qtree ever has an ACL.

In the first two modes, the filer acts exactly as it would if it were an NFS or Windows server. The other two modes, which involve access to a file by a nonnative client, are more complex and are discussed at length in the following sections.

4.1 Share-Level ACLs

All shares exported by a NetApp filer to CIFS clients have special ACLs set on them called *share-level ACLs*. By default, these allow the special Windows group "everyone" full access, which means that no additional restrictions are imposed at the share level on what users can do to the files in the share. System administrators can elect to set more restrictive ACLs on shares, however. When this happens, all operations coming in from CIFS clients are first checked against the share's ACL to decide whether they can progress farther (this is done anyway, but because the default ACL grants everyone full access, it is as if no test were done). This check is not always mentioned in the following descriptions of how files are authenticated, but it is important to remember that it is always done first, before any of the other permissions are checked.

The concept of share-level ACLs does not apply to clients using the NFS protocol to access files (UNIX or (PC) NFS clients), as those are accessed through NFS-exported directories. The export settings of the exported directory serve a similar purpose in the NFS world.

4.2 The Qtree Security Style

Data ONTAP implementation has the idea of a "qtree," which is an extension of the quota tree concept in previous releases of the OS. Briefly, a qtree can have its security style set to "UNIX," "NTFS," or "mixed." The default security style for a qtree is inherited from the security style set on the volume, which is set with the qtree command just as if the volume (e.g., */vol/vol0*) is itself a qtree.

It is a common misconception that the security style of a qtree controls who can access a file system. The truth is that qtree security type controls the type of permissions applied to files and directories in that file system. In other words, it controls who can apply security to a file, not *who can access a file. Therefore, you* do not *need to have a mixed or NTFS style qtree for PC users to access data on a filer. Similarly, you* do not *need to have a mixed or UNIX style qtree for UNIX users to access data on a filer.*

Files in UNIX qtrees can have ACLs set on them, because qtree styles may be changed (from NTFS to UNIX) after files in them have already been created and, perhaps, modified. If an ACL does exist on such a file, it is ignored by all authentication and permissions-getting operations. Any subsequent change in permissions to a file results in removal of the ACL, if it exists, and the resulting permissions are always "true" UNIX permissions. In other words, it is not possible to set or change an ACL on a file in a UNIX qtree.

Similarly, files in NTFS qtrees can have genuine UNIX permissions, but any change to the security style results in setting an ACL on them, and subsequent accesses by UNIX clients are validated using the UID-to-SID mapping mechanism and the file's ACL. It is not possible to directly set the UNIX permissions on a file in an NTFS qtree.

Files in mixed qtrees can have either security style; they always have the security style associated with the client that was last used to change their access permissions or ownership. When a UNIX client sets the permissions on a file, any ACL on it is removed. When a PC client sets the permissions on a UNIX file, an ACL is constructed and attached to the file, which becomes a PC file. A set of synthesized UNIX permissions is also generated at that time for use by UNIX clients.

Because of the conservative mapping used by NetApp, changing the security style on a file from UNIX to PC and back—by setting an ACL, and then setting UNIX permissions that match the synthesized permissions calculated from the ACL—can result in a net permissions change. This unintuitive behavior occurs whenever there is loss of information going from one security style to the other. For example, if an ACL allows access to everyone but denies it to user `agy`, the synthesized UNIX permissions cannot allow access to everyone. In some cases, enterprise customers have various administrators on both UNIX and NTFS side changing file permissions on a frequent basis. *It is not uncommon that the administrators from Windows and UNIX world sometimes run into each other and mess up each other's permissions. For these customers, it is recommended not to use mixed qtrees unless it is absolutely necessary.* UNIX shops that use groups heavily, however, could also have problems with mixed permissions. This is because the OS has no way of determining what UNIX groups a mapped CIFS user might belong to, so the permissions for the UNIX group of the file are set to the same as the permissions for others.

Loss of information does *not* happen when you change the security style of a qtree. Changing the security style of a qtree does not affect the files in the tree, so you can change a qtree from NTFS style to UNIX style, and back, without changing permissions on any of the files in the qtree. This is true even if the file system has been dumped and restored (using the NetApp `dump` and `restore` console commands) in the meantime. However, just as with mixed style security, any permission changes made subsequent to the qtree security style change will become the effective security and will replace the style of security settings that were in place before the security style change.

Either UNIX or PC clients can access files in a qtree. As was said in the last section, file accesses when the file and client type are the same have the same semantics as if the client were accessing a file on its native file system. The cases where the client and file security style do not agree are discussed next.

4.3 Access to a File without an ACL by a PC Client

When a PC client requests access to a file without an ACL, the UNIX permissions on the file are used to determine whether to grant access to the file. The requester's NTFS SID is looked up on the domain controller (or local user database) to get a username. The username is then mapped to a UNIX username using the name-mapping feature, which is then looked up through the */etc/passwd* file (or NIS or LDAP) to get the mapped user's ID and group IDs (UID and GIDs). These are then compared to the UID and GID and associated permissions for the file, exactly as if the requester were the mapped UNIX user.

If the client wants to *set* the permissions on a file, the result depends on the qtree security style. If the file is in a UNIX qtree, the PC client cannot set an ACL on the file and can perform only the actions allowed by its mapped UNIX ID. This means that the PC client must use a remote session on a UNIX client to set the UNIX permissions on the file. These functions are described as follows in Figure 5 and table:

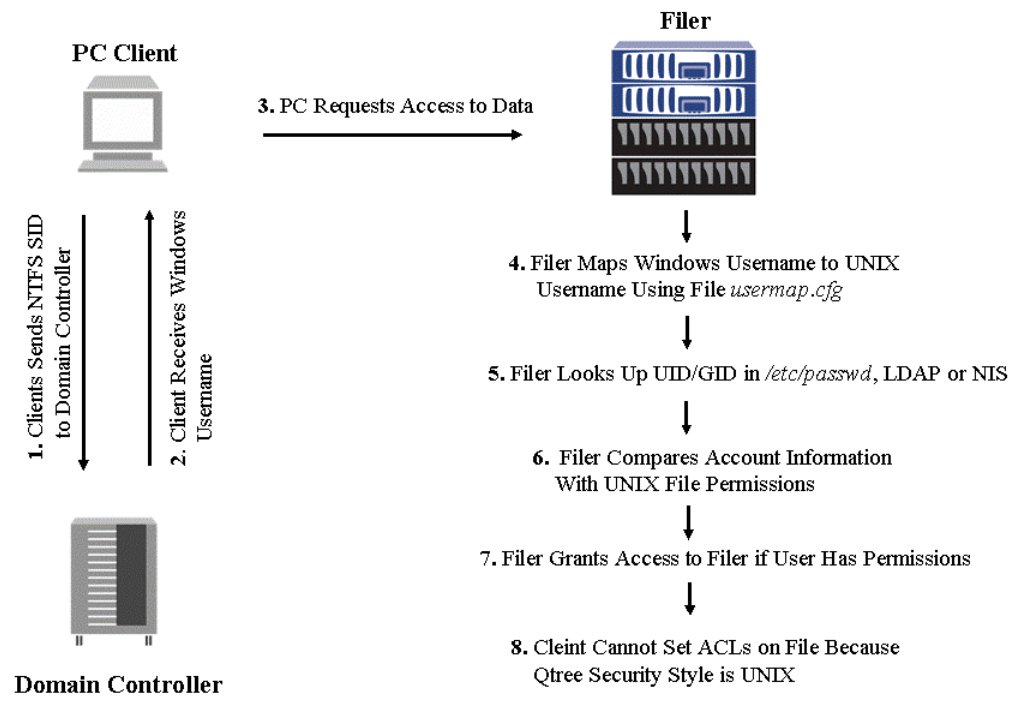

PC Client
Filer
3. PC Requests Access to Data
1. Clients Sends NTFS SID to Domain Controller
2. Client Receives Windows Username
4. Filer Maps Windows Username to UNIX Username Using File *usermap.cfg*
5. Filer Looks Up UID/GID in */etc/passwd*, LDAP or NIS
6. Filer Compares Account Information With UNIX File Permissions
7. Filer Grants Access to Filer if User Has Permissions
8. Cleint Cannot Set ACLs on File Because Qtree Security Style is UNIX
Domain Controller

Figure 5) Access to a file without an ACL by a PC client.

Current Permissions Type	The File Is a Directory	The File Is a Normal File
UNIX	Set an ACL on the directory. Permission to do this must have been granted by the permissions gotten from WAFL on the original access request. The new ACL overrides any previous UNIX permissions and is used to determine whether any future attempts to add, delete, or change files in the directory should succeed. This case would be valid if the qtree style was mixed or if the qtree style had previously been UNIX but was changed to NTFS or mixed.	Set an ACL on the file. Same basic rules as for directories.
NTFS	Replace the current ACL. Permission to do this must	Replace the

	be granted by the current ACL before such an operation can succeed. The file's owner can always do this, and administrators can always take ownership of a directory and then set the ACL. This case would be valid if the qtree security style is mixed and the security style of the directory or file is already NTFS style or if the qtree security was NTFS.	current ACL. Same rules as for directories.

4.4 Access to a File with an ACL by a UNIX Client

In a UNIX qtree, a file can have an ACL, but the ACL is ignored and standard UNIX permissions checking rules are used.

In a mixed qtree, when a UNIX client requests access to a file with an ACL, the ACL was formerly translated into a set of “effective” UNIX permissions that the client can understand. The UID-to-SID mapping mechanism is now used to do permissions checking, but the translation is still done, for use by some UNIX operations such as `chown`, for when CIFS is unavailable or when the qtree style is changed, and so forth. The same “conservative mapping” rule as mentioned previously is used in doing the translation, so that no UNIX user can get access to a file that he or she could not have obtained under the CIFS protocol. This mapping is done at the time you set the ACL, so the effective permissions are always available for authenticating UNIX clients. If the UNIX client subsequently sets the permissions on the file, however, the resulting permissions are “true” permissions, and the ACL is removed.

If the file is in an NTFS qtree, file access works the same way as in a mixed tree, but the UNIX client cannot set UNIX permissions on the file. NetApp does not presently furnish a way for UNIX users to set ACLs on files. These functions are described below in Figure 6.

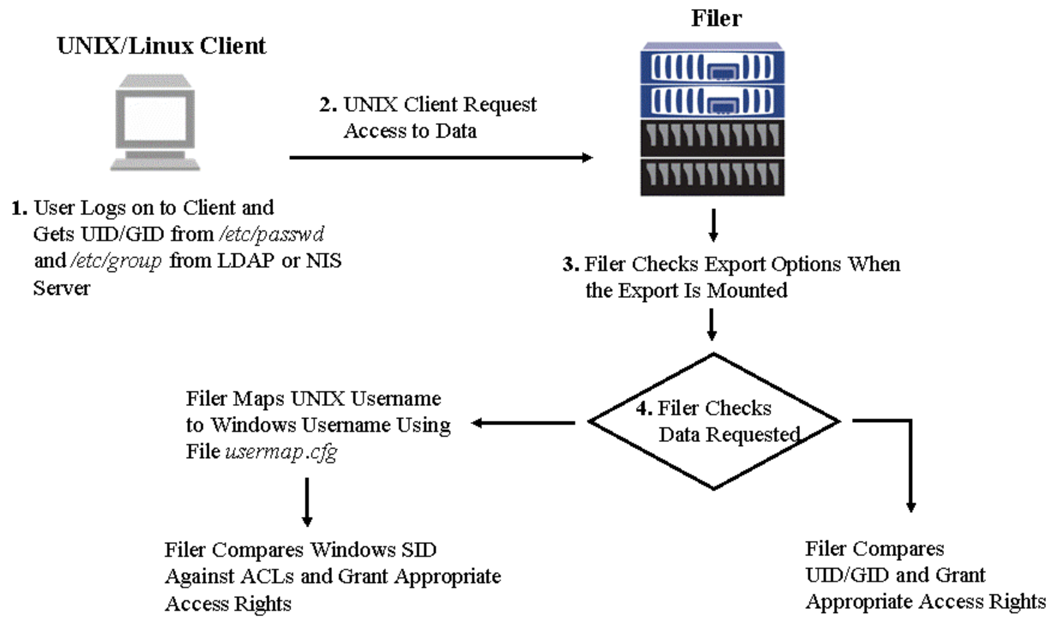

UNIX/Linux Client
Filer
2. UNIX Client Request Access to Data
1. User Logs on to Client and Gets UID/GID from /etc/passwd and /etc/group from LDAP or NIS Server
3. Filer Checks Export Options When the Export Is Mounted
4. Filer Checks Data Requested
Filer Maps UNIX Username to Windows Username Using File usermap.cfg
Filer Compares Windows SID Against ACLs and Grant Appropriate Access Rights
Filer Compares UID/GID and Grant Appropriate Access Rights

Figure 6) Access to a file with an ACL by a UNIX client.

5. Another Data Access Protocol—HTTP

NetApp support of HTTP is similar to support for NFS and CIFS in that each request is serviced by a continuous process, from the network driver through the file system and back. Having the HTTP server embedded inside the Data ONTAP real-time microkernel offers a significant performance advantage over traditional Web servers that run as applications in user space, outside the kernel on UNIX or Windows platforms.

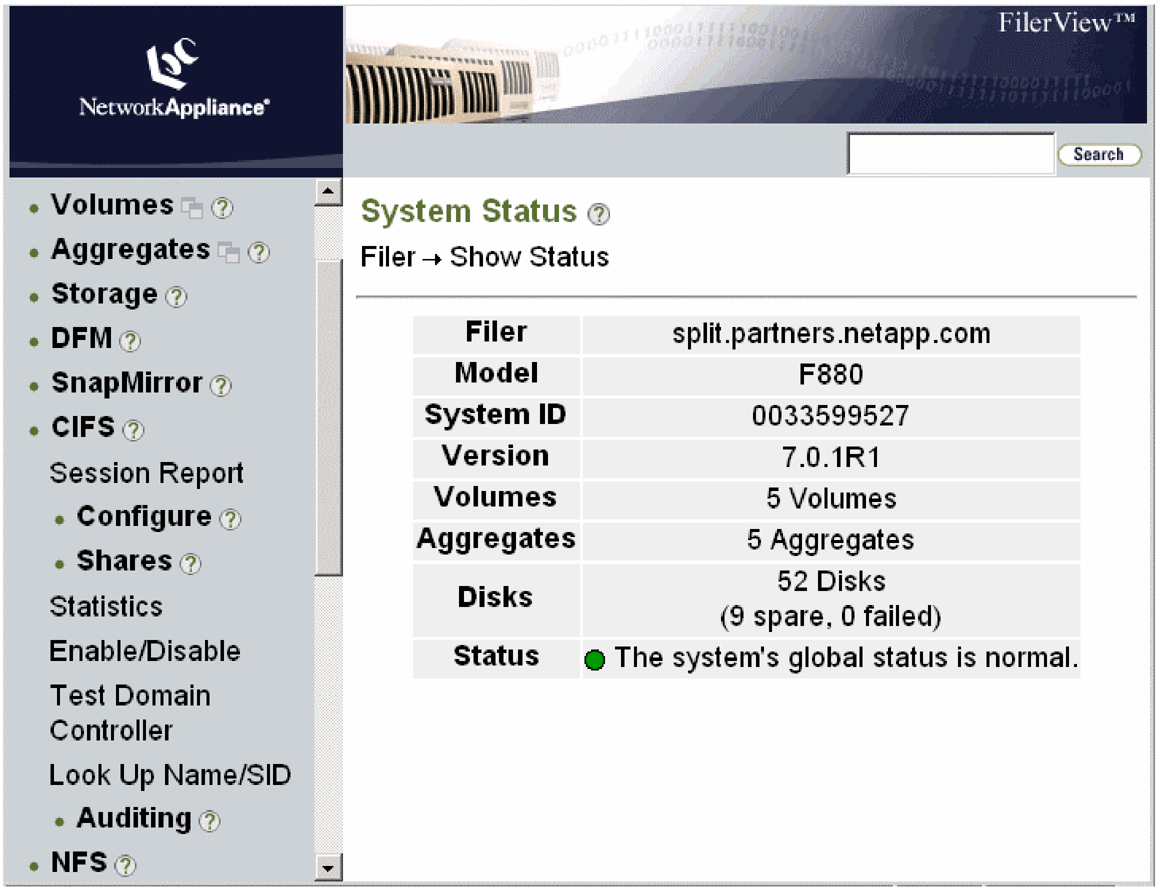
In the current implementation of NetApp Web Filer™, only the HTTP `get` operation is supported. This means that Web Filer may need to be deployed in concert with one or more traditional web servers, to handle any non-`get` operations. Typically, `get` operations account for the overwhelming majority of HTTP requests received by a Web server, and so by handling the lion's share of the incoming load, Web Filer can have a dramatic impact on the effective performance of a Web site. Furthermore, the companion Web servers can (and *should*, for simplest administration and most scalable configurability) be configured as NFS or CIFS clients served by the filer, facilitating the use of a consistent file naming convention within a common file system across both (or all) Web servers.

For further discussion of Web site configuration using Web Filer, see NetApp Technical Report, "Migrating to a Web Filer" [TR-3012]. For other even more effective methods of scaling Web service, NetApp Net Cache® products can be used in conjunction with (or instead of) Web Filer.

5.1 HTTP for Filer Administration

The GUI provided for administration of a NetApp filer uses HTTP. This allows the user to sit at any platform and use any Web browser to execute administrative tasks. This does not require the HTTP protocol to be licensed and enabled on the filer, because it uses a special, nonstandard port and facilitates access only to specific administration files and actions.

Figure 7 provides an example of FilerView®, the HTTP-based GUI for NetApp filer administration.



System Status

Filer → Show Status

Filer	split.partners.netapp.com
Model	F880
System ID	0033599527
Version	7.0.1R1
Volumes	5 Volumes
Aggregates	5 Aggregates
Disks	52 Disks (9 spare, 0 failed)
Status	● The system's global status is normal.

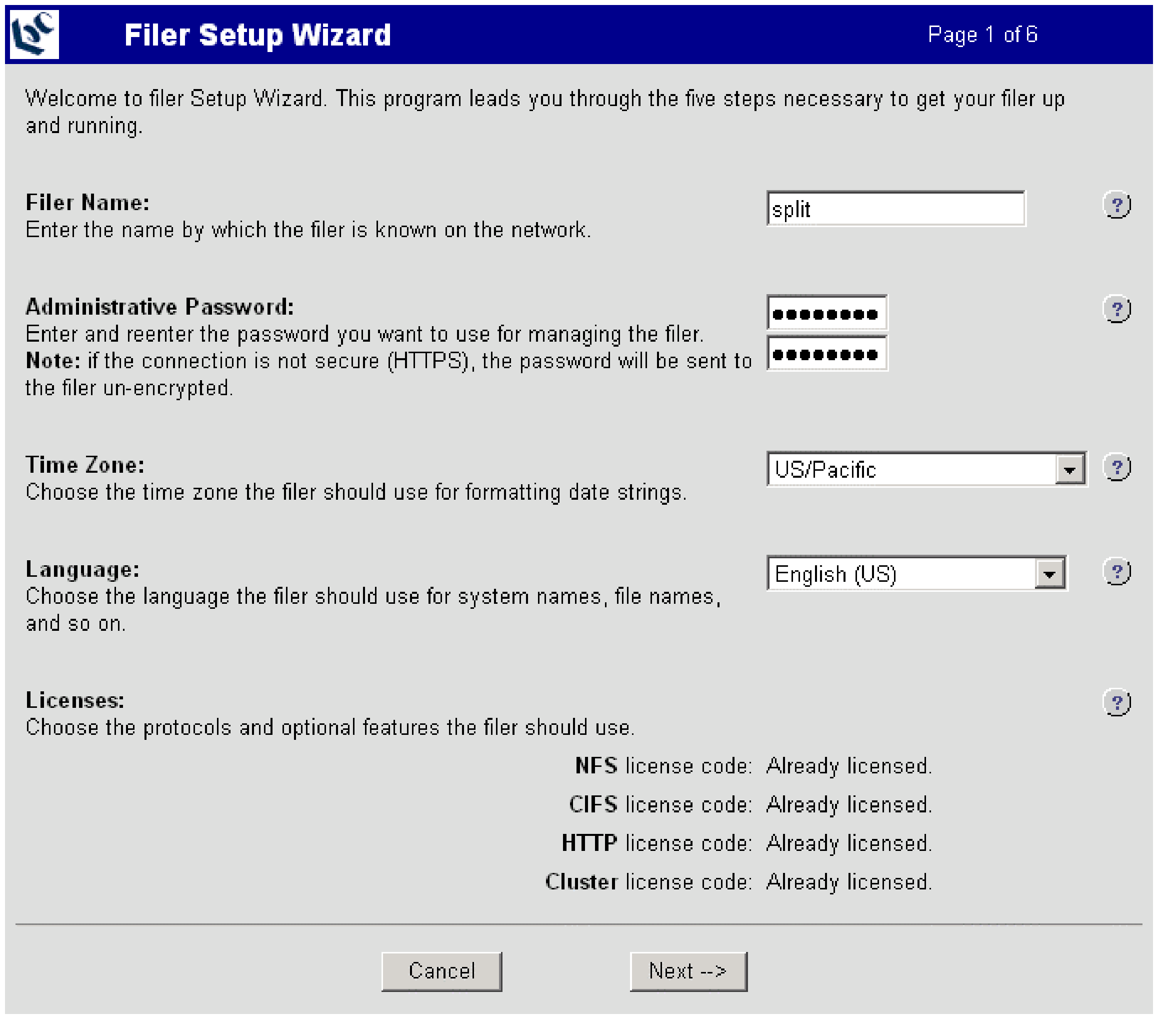
Figure 7) FilerView.

You can use this tool to monitor the filer's status remotely. Performance is displayed graphically. You can control configurable parameters and options (e.g., setting the `minra` option for minimum read-ahead). And you can apply all actions that change the filer's current, active configuration to configuration files that the filer consults at startup time. In a future release, simultaneous administration of multiple filers will be added.

For additional information about HTTP-based GUI administration tools for NetApp filers, see the NetApp System Administrator's Guide on the NOW™ (NetApp on the Web) site.

5.2 HTTP for Filer Setup

The Filer Setup Wizard is designed to facilitate setup of a filer. As is the case with FilerView, Filer Setup Wizard can be run by a Web browser on any platform and doesn't require that the HTTP protocol be licensed. The wizard is used to set up protocol licenses, current time zone, language options and network information. See Figure 8 for an example of Filer Setup Wizard.



Filer Setup Wizard Page 1 of 6

Welcome to filer Setup Wizard. This program leads you through the five steps necessary to get your filer up and running.

Filer Name:
Enter the name by which the filer is known on the network. split

Administrative Password:
Enter and reenter the password you want to use for managing the filer.
Note: if the connection is not secure (HTTPS), the password will be sent to the filer un-encrypted. ●●●●●●●● ●●●●●●●●

Time Zone:
Choose the time zone the filer should use for formatting date strings. US/Pacific

Language:
Choose the language the filer should use for system names, file names, and so on. English (US)

Licenses:
Choose the protocols and optional features the filer should use.

NFS license code:	Already licensed.
CIFS license code:	Already licensed.
HTTP license code:	Already licensed.
Cluster license code:	Already licensed.

Figure 8) Filer setup using HTTP.

For additional information about HTTP-based GUI administration tools for NetApp filers, see the NetApp System Administrator's Guide (available online).

6. Using Multiprotocol Filers

There are many protocol combinations available on NetApp filers: NFS-only, CIFS-only, and NFS-plus-CIFS. In all three cases, HTTP access can also be enabled. Note that *NFS* here implies both NFS versions 2 and 3, operating over UDP and/or TCP transport layers (see Figure 3). For more information about the configuration options and software offerings, see the NetApp Software Configuration Guide available at the NOW site.

Multiprotocol filers are designed to make deployment into heterogeneous (mixed NFS and CIFS) environments as easy as possible. The homogeneous (all NFS, or all CIFS) cases are simpler to plan and implement. For the purposes of this paper, the discussion assumes a mixed environment with UNIX-based NFS clients as well as Windows-based CIFS clients.

6.1 Administration and Security

- In general, whether in the context of NFS or CIFS, file servers must constrain access to files based on file ownership, explicit permissions (e.g., read and/or write privileges) granted to other users, implicit permissions inherited from the local directories in which the files are found, or the directory tree as exported or shared by the file server.

- File access restrictions are typically created by users (for the files they own) and systems administrators (for shared file systems and directory trees, or when creating a user's home directory).

- User (and sometimes client system) identification must be authenticated prior to file access.

- All of the above are different for NFS and CIFS. Both NFS and CIFS methods are explained and compared in Sections 6.1.1 and 6.1.2, below. Section 6.1.3 describes how these security models are comprehensively merged on NetApp filers.

In a heterogeneous environment, information about users—usernames, group affiliation, and passwords—is accessible in multiple formats from multiple sources. Merging the namespace for users is strategically important to the task of administering shared resources on a network. This is usually accomplished by assigning UNIX-style UIDs and GIDs (integer-value User IDs and Group IDs) to PC users. Thereafter, the database of PC users must be carefully updated in two places: typically via NIS (Network Information System) among the UNIX platforms, and via Windows NT/AD (Active Directory) Domain on the PCs. This is onerous, especially for sites with large numbers of Windows users that do not have any other need for having a UNIX UID and GID assigned to them. (NetApp offers two methods for eliminating this time-consuming administrative overhead, as described below in Section 6.1.3.)

User information is used by a NetApp filer to enforce security—basically for authentication of the user and to regulate file access based on permissions. These mechanisms and events are fundamentally different for UNIX and NFS versus PCs and CIFS. First, let's look at NFS and then contrast it with CIFS.

6.1.1 NFS

- NFS exports data to a client *system*. NFS relies on the client to authenticate its own users. Given the complete local control a user might have over a client system, the NFS security model can be subverted relatively easily.

- NFS is UNIX-derived and therefore implements a UNIX-style set of file permissions limited to three categories of users (user/group/other) and three types of file permissions (read/write/execute).

- An NFS file server can apply various options (e.g., restricting all access to *read only*) when exporting, and an NFS client has similar options when mounting from an NFS file server;

- A NetApp filer can export to specified NFS clients, netgroups (lists of clients), or even network subnets (note that exporting by subnet is not always available on conventional NFS file servers from other vendors).

With UNIX NFS clients, file systems are "mounted" from the server so as to be logically indistinguishable from local file systems. This can happen without any user activity—simply as a result of starting up the client, for example. The NFS server will grant or deny the mount request based on the hostname of the client, as dictated by restrictions specified in the */etc/exports* file (which contains lists of hostnames—or on a NetApp filer it could also be a list of network subnets). No password or other authentication is required by an NFS server[7].

User authentication is performed by the client, *not* the server. The client-authenticated user's UID must be passed across the network with all NFS requests sent to the server.

[7] (PC) NFS requires a username and password at mount time, but that is to rectify a deficiency, from the UNIX perspective, in the PC client, and not a function of the NFS protocol *per se*.

Therefore, if a user can convince[8] the NFS *client* of identity *username* associated with the UID *n*, then an NFS server will accept that any NFS requests issued by that client on that user's behalf are for UID *n*. One exception to this occurs when *n* corresponds to the superuser ("root"), in which case an NFS server will assign unauthenticated status (the user "nobody")—unless root access privileges have been assigned to that client in the */etc/exports* file. It contains the list of resources to be exported when Data ONTAP starts up, restarts, or receives the `exportfs -a` command (given that a valid NFS license exists). Maintaining the */etc/exports* file enables resources to be available automatically upon filer startup and eliminates the need to enter the `exportfs` command for each resource you want to export.. In the */etc/exports* file, you also specify the access restrictions and security service with which clients can mount each resource. At any time after filer startup, you can override specifications in the */etc/exports* file by using the `exportfs` command (until the filer is restarted or another overriding `exportfs` command is issued).

Each time an export rule is loaded into memory, it undergoes a straightforward conversion from a string to an entry in a hash table. The basic string format is:

```
/vol/volFoo -rw=host1,ro=host2
```

This specifies that host1 has read/write access on volume volFoo on the filer while host2 has only read-only access right. However, if `-rw` or `-ro` is specified with no hosts, for example:

```
/vol/volFoo -rw=host1,ro
```

then it implies all hosts. So the above example would be host1 gets read/write, and all other hosts get read-only.

Note that the current version of Data ONTAP supports user authentication through Kerberos. By specifying security services option "sec=krb5" in the exports list, you can control NFS/CIFS access to a specific volume at the user lever. For more information on Kerberos, refer to TR3367 and TR3085.

6.1.2 CIFS

- CIFS shares data with *users* (not machines, as with NFS). Authentication of a CIFS user's identity is done by the server, not the client (as with NFS).

- In a Windows NT/AD domain, the server can use another system (the domain controller) for authentication of users' credentials. It is easier to make file

[8] This can be done with a valid password (perhaps one that has been guessed or stolen), or by some means that bypasses the use of the password on the client (e.g., using an NFS file handle snooped from legitimate network traffic).

server(s) and domain controller(s)[9] secure from tampering than to make *all* machines on the network secure (as must be done for NFS to be used safely). Refer to TR3362 on how to join the filer to Active Directory domain.

- CIFS share, directory, and file permissions are controlled by ACLs. ACLs can include permissions specific to individuals or groups of users.

- Users accessing files via CIFS are generally more at risk when the file server restarts than would be the case with NFS. Therefore, CIFS users should be warned well in advance of file server restarts. (NetApp filers provide automatic notification of file server shutdowns and restarts, sent to all users with open CIFS sessions.)

A CIFS server provides access to a file system (or a directory tree portion of a file system) in a unit called a *share*. In many respects this is similar to an NFS export, except that NFS exports file access to a remote *host* (essentially, an IP address), whereas CIFS shares are network accessible to *users*, not machines.

Unlike NFS, no UID is passed over the network with each CIFS request. Nor does the client "mount a file system" without an active user (as does a UNIX NFS client). Instead, the initial connection-establishment sequence requires the user's PC to provide credentials to the file server. In a Windows domain, these credentials were previously obtained when the user first logged on to the domain (when NetLogon required the user to provide a password). The server then authenticates the user's credentials (either with a domain controller through Active Directory or by looking up the user in a local file), and obtains additional information (user ID and group affiliation) used for the duration of the connection. (See Section 6.4 for more about authentication with NetApp filers.)

Notice the higher granularity and heightened security of CIFS compared to NFS. Each CIFS *session* provides a single[10] specific user with access to a single specific share, with access authenticated by the server, not the client. This requires a CIFS server to maintain a persistent state, whereas an NFS server is stateless, trusting the client for user authentication and relying on it to provide information (e.g., UID and GID) repeatedly, with every request.

The downside to this is resiliency after failure. If the CIFS server restarts, connections are broken and sessions are lost, requiring the user's PC to repeat the authentication procedure (i.e., supply a username and password in a dialog with the server—usually executed automatically and transparently from the user's perspective). With some applications running on a CIFS client (e.g., Microsoft WordPad), a server restarting very

[9] There can be multiple domain controllers (DCs) for redundancy purposes.

[10] Multiple-user Windows server implementations (for example, WinDD [Tek-96] and WinCenter [NCD-96]) can establish CIFS sessions for many users.

quickly[11] could conceivably reestablish the connection without any impact on the user, as long as no attempt is made to save an open file before the server has completed restarting. Other applications might attempt to perform I/O to the server more continuously, leaving no window for client-tolerable server restarting. If a CIFS client attempts file access on an established connection while the server is unavailable (down or not yet finished restarting), this is effectively the equivalent of a failed disk from the perspective of the application software. In many cases, the application will report an error and allow the user to retry, but some applications will simply hang or exit.

Unlike other CIFS implementations, NetApp filers can provide automatic notification of shutdowns and restarts, sent to all users with open CIFS sessions whose machines have been configured to receive such messages. This will at least give users an opportunity to save their work (and in some cases, exit the application if necessary), without the system administrator needing to remember to broadcast a warning.

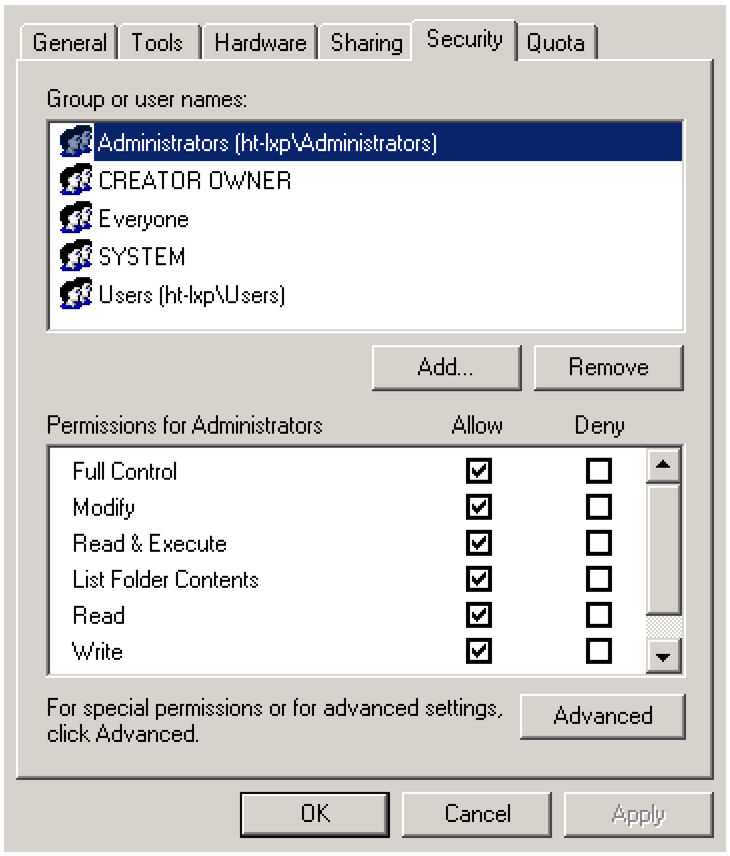
NFS's statelessness allows it to tolerate events like restarts without impact on the user's application (which simply waits on the client side, under most circumstances, until the server has restarted). Essentially, NFS was designed such that each operation would be independent, but CIFS is session-oriented. There are pros and cons to both approaches, but it must be conceded that CIFS offers a stronger security model than NFS.

File locking (see Section 6.5)—a component of security in some contexts—is also stronger with CIFS than with NFS.

Another important difference between NFS and CIFS is the nature of the file permissions themselves. Instead of the limited file permission model[12] in UNIX and NFS, NTFS and CIFS permits a richer file access permission description. ACLs define a list of users and/or groups mapped to permissions, and can be as long as necessary. Figure 9 offers an example of an ACL. See Section 2.2.1 for more details.

[11] NetApp filers reboot in about one minute, regardless of the size of the file system or the configuration of the server.

[12] Although some versions of UNIX also support ACLs, these ACLs are not the same as CIFS ACLs, and are not yet interoperable among diverse UNIX implementations.



Group or user names:

- Administrators (ht-lxp\Administrators)
- CREATOR OWNER
- Everyone
- SYSTEM
- Users (ht-lxp\Users)

Permissions for Administrators	Allow	Deny
Full Control	☑	☐
Modify	☑	☐
Read & Execute	☑	☐
List Folder Contents	☑	☐
Read	☑	☐
Write	☑	☐

For special permissions or for advanced settings, click Advanced.

Figure 9) ACLs in NTFS.

A user's access privileges for a file are determined by the user's explicit presence in the ACL, or membership in a group included in the ACL. An ACL entry can define rights granted or explicitly deny all access. If a user's multiple group affiliations result in an ACL both granting *and* denying permission for one or more forms of file access, then denials override grants.

As with options applied to the whole exported (or mounted) file system (or exported/mounted subtree) with UNIX and NFS, CIFS shares also have global properties—an ACL that applies to the whole share. In the event that multiple shares overlap, and include one or more directories in common, the applicable ACL derives from the share that provided the client's connection path to the file.

6.1.3 Multiprotocol Filer

- Tree-by-tree, system administrators can configure a NetApp filer to implement PC-style file access permissions (which apply administrator- or user-specified ACLs), UNIX-style permissions (read/write/execute), or "mixed" mode where both styles of file access permissions are applied.

- NetApp filers offer the industry's only *completely* integrated locking across protocols (see Section 6.5).

- The Active Directory Users and Computers snap-in (Windows Server Manager and User Manager utilities for Windows NT), or the Computer Management snap-in can be used to administer NetApp multiprotocol filers. Also, NetApp offers FilerView for an administrative interface via a Web browser, or a command-line administrative interface can be used via telnet or the console serial port.

- NetApp filers resolve UNIX-style hard and symbolic *links* (conceptually similar to Windows *shortcuts*) on behalf of Windows users, when the links point elsewhere within the filer. (Hard links always meet this requirement, but symbolic links can point anywhere.)

- Windows NTFS based file access auditing is available for both Windows (CIFS) and UNIX (NFS) clients.

A NetApp multiprotocol filer is designed to meet the administration and security requirements of a *heterogeneous* environment. The applicable requirements are somewhat different than for a server intended for a purely UNIX or purely PC network. Fundamental differences in file attributes (especially access permissions, file names), authentication, locking, and statefulness/statelessness, make this particularly challenging. For all of the above considerations, the features and behavior of the NetApp filer have been extended *natively* to provide file service to all users in the mode that is familiar for each of them, with no trade-offs in performance, reliability, or ease of administration.

NetApp has implemented NTFS-style file-, directory- and share-level ACLs by extending the WAFL file system (which has the advantage of being neither a UFS nor an NTFS). Therefore, unlike emulated CIFS file service applications running on competitors' UNIX-derived file servers, NetApp does not need to maintain a "shadow database" of files containing the NTFS-style ACLs. This has several benefits:

- Faster (lower latency, faster response time) file access
- Stronger integrity in the event of an unplanned shutdown (i.e., no possibility for the actual file system and a "shadow database" to get out of sync)
- UNIX file access metadata is accessible to Windows clients as NTFS-style "extended file attributes"
- UNIX-style ACLs will be available with the emerging NFSv4 standard.

There are three modes of file access enforcement modes available[13] on a NetApp multiprotocol filer: UNIX-style; NTFS-style; and mixed mode. The mode can be specified for the whole filer or for individual qtrees . A qtree has other properties as well, most important being a "soft partition" disk space quota, which can be modified on-the-fly. [TR-3002]; for additional information about qtrees, see the *System Administrator's Guide* (available online).

In UNIX-style mode, UNIX security semantics are employed. All CIFS file accesses will be primarily constrained by each file's mode bits (rwx for u/g/o). Windows users are either mapped statically or on-the-fly by the filer to equivalent UNIX namespace UIDs (see below).

In NTFS-style mode, NTFS-style ACLs define file access permission on a file-, directory-, or share-level basis. By default, each file or directory inherits the ACLs of its parent.

In mixed mode, both NTFS-style and UNIX-style permissions are used. This is the most widely used approach on NetApp multiprotocol filers at customer sites.

Additional details and examples are provided in *File System Security: Secure Network Data Sharing for NT and UNIX.*

Currently, only user accounts are mapped between UNIX and Windows; group to group mapping (GID <=> SID) is not part of the NetApp multiprotocol solution. Though this functionality could be added to Data ONTAP, we feel that it is not only unnecessary but also adds a level of complication. Since the mapped user is a member of all the groups that would be associated with it when logging in natively, as long as both sides' group membership is maintained, the effect is the same as if the groups were mapped. For further details on this, see the document "Frequently Asked Questions about Security on NetApp Filers."[14]

Symbolic link files (symlinks) are another multiprotocol challenge in mixed PC and UNIX environments. These files are created by UNIX machines and contain the pathname of another file, such that any access of a symlink will automatically be redirected to the other file. (This is similar to the behavior of a Windows shortcut file.) NetApp filers correctly interpret the content of symlink files on behalf of CIFS users when the pathname resolves to a file elsewhere on the same NetApp filer.

---

[13] The system administrator defines the applicable mode(s) for the filer as a whole or for individual trees within the file system during initial setup. The file access security model can also be redefined on-the-fly at a later date.

[14] Please see "Frequently Asked Questions about Security on NetApp Filers." This document resides on NetApp's NOW™ (NetApp on the Web) site (now.netapp.com). Access to this site is free to all NetApp customers but requires a login name and password.

The WAFL file system can be set to track file access by both NFS and CIFS clients. The security log is written in the same format as the native Windows security log and thus can be viewed with the Windows Event Viewer.

6.1.4 Data ONTAP Security Options

Some options in Data ONTAP are related with security setting and are described as follows:

- **`cifs.perm_check_use_gid`**
  This option affects security checking for Windows clients of files with UNIX security where the requestor is not the file owner. In all cases Windows client requests are checked against the share-level ACL, then if the requestor is the owner, the user permissions are used to determine the access.

  If the requestor is not the owner and if `perm_check_use_gid` is "on" it means files with UNIX security are checked using normal UNIX rules, i.e., if the requestor is a member of the file's owning group the "group" permissions are used, otherwise the "other" permissions are used.

  If the requestor is not the owner and if `perm_check_use_gid` is "off," files with UNIX security style are checked in a way that works better when controlling access via share-level ACLs. In that case the requestor's desired access is checked against the file's "group" permissions, and the "other" permissions are ignored. In effect, the "group" permissions are used as if the Windows client were always a member of the file's owning group, and the "other" permissions are never used.

  The default setting is "on" for new installations. For existing installations, this has the opposite effect of the old "PC-mode" installation setting.

  If you do not plan to use share-level ACLs to control access to UNIX security style files (e.g., in a UNIX qtree), you might wish to change this setting to "on."

- **`wafl.default_unix_user`**
  Specifies the UNIX user account to use when an NT user attempts to log in and that NT user would not otherwise be mapped. If this option is set blank, such accesses will be denied.

- **`wafl.root_only_chown`**
  When enabled, only the root user can change the owner of a file. When disabled, nonroot users can change the owner of files that they own. When a nonroot user changes the owner of a file they own, both the set-UID and set-GID bits of that file are cleared for security reasons. A nonroot user is not allowed to give away a file if it would make the recipient overrun its user quota.
  `wafl.root_only_chown` is enabled by default.

- **`wafl.default_nt_user`**
  Specifies the NT user account to use when a UNIX user accesses a file with NT security (has an ACL), and that UNIX user would not otherwise be mapped. If this option is set to the null string, such accesses will be denied. The default value for this option is the null string.

- **`wafl.default_security_style`**
  Specifies the default security style assigned to a new volume. All qtrees created on the volume get this as their security style. Legal values for this option are `unix, ntfs,` or `mixed.` The default value for this option is `unix,` unless the filer is an NTFS-only filer, in which case the default is `ntfs.`

- **`wafl.default_unix_user`**
  Specifies the UNIX user account to use when an authenticated NT user did not match an entry in the *usermap.cfg* file. If this option is set to the null string, NT users that are not matched in the *usermap.cfg* file will not be allowed to log in. The default value for this option is `pcuser.`

- **`wafl.nt_admin_priv_map_to_root`**
  When on (the default), an NT administrator is mapped to UNIX root.

- **`wafl.root_only_chown`**
  When enabled, only the root user can change the owner of a file. When disabled, nonroot users can change the owner of files that they own. When a nonroot user changes the owner of a file they own, both the set-UID and set-GID bits of that file are cleared for security reasons. A nonroot user is not allowed to give away a file if it would make the recipient overrun its user quota. `wafl.root_only_chown` is enabled by default.

6.2 DNS and WINS

- Rather than searching local host tables (files containing lists of hostnames with corresponding IP addresses), IP address resolution is typically done using DNS among UNIX systems and with Windows 2000 and 2003 domains. In Windows NT domains, IP address resolution is done with WINS.

- In Windows 2000 and 2003 domains running in native mode, WINS is no longer necessary; however, when running in mixed mode or when on an NT4 domain, WINS is essential.

- The conventional BIND (prior to version 9) based DNS name resolution is *static* (based on text files that must be updated manually by system or network administrators). WINS and DDNS (Dynamic DNS, used in Windows 2000, 2003, and XP and supported by BIND version 9) is *dynamic* (updated automatically without human intervention).

- NetApp filers can participate in DNS, WINS. CIFS DDNS supported was added in Data ONTAP 7.1.

PCs and UNIX systems have historically used different mechanisms for name resolution—acquiring the IP address for a system based on its name.

UNIX systems resolve IP addresses via DNS. DNS is essentially a remote table-lookup mechanism. Given a "fully-qualified" hostname—*[hostname].[subdomain].[top-level-domain]* (e.g., *www.netapp.com*), DNS returns the IP address for it. Changes to addresses (or new addresses) are maintained by editing host table files on DNS servers. UNIX systems send queries (with fully qualified hostnames) to DNS servers and receive IP addresses in the replies.

In addition to DNS, PCs can use WINS (Windows Internet Naming Service)[15]. WINS is similar to DNS except that it maps NetBIOS names to IP addresses rather than fully qualified domain names. When a PC starts up, if configured for WINS it will engage in a dialogue with the WINS server, registering its current NetBIOS hostname and IP address. Please note that in Windows 2000, 2003, and XP, DDNS is added and WINS is no longer needed.

When network resources are sought out (e.g., while browsing the Network Neighborhood[16] looking for CIFS shares) the PC client receives the most current information from either the WINS or DNS server about other hostname IP addresses. If these remote machines should be restarted with new addresses, the WINS (and DDNS) servers will be notified and their address maps updated automatically, with none of the manual steps required by static DNS. With DDNS WINS, no files need to be edited to include new addresses.

Dynamic address resolution reflects the intent of PC developers to simplify the management of the IP address space for a network. It also makes possible the sequence of events by which a CIFS client finds a server and accesses a share. Unlike NFS, which requires foreknowledge of available servers and their file systems (and creation of suitable mount points on the client), CIFS shares can be discovered by browsing. Therefore, even if the CIFS server happens to have a different address today than it did yesterday, the client can still find it in the Network Neighborhood.

[15] WINS was previously known as NBNS (NetBIOS Name Service) for NBT (NetBIOS-over-TCP/IP). WINS is no longer needed in a pure Windows 2000 or 2003 server and XP workstation environment.

[16] "Network Neighborhood" is Microsoft's name for the Windows GUI that displays available network resources.

A NetApp multiprotocol filer can be configured to participate in both DNS (or DDNS) and WINS. In fact, a NetApp filer can be configured with multiple NetBIOS names, allowing it to appear to be more than one CIFS server in the Network Neighborhood.

6.3 Using the *Usermap* File

By default, Windows names map to identical user names in the UNIX space; for example, the Windows user `agy` maps to the UNIX user `agy`. You can use the user mapping file */etc/usermap.cfg* to map PC users to UNIX users that are named differently and to handle various special or generic users such as root, guest, administrator, nobody, and so on. In Data ONTAP, legal entries in the file are as follows:

Windows_username	UNIX_username
Domainname\username	UNIX_username
Windows_username	“”
*	UNIX_username

The default mapping is bidirectional between Windows and UNIX. The Windows name, which can optionally be qualified by the domain it belongs to, is mapped to the UNIX username given in the right column and vice versa. For example, the mapping

```
johndoe jcd
```

maps a Windows user named johndoe to the UNIX user jcd and also maps UNIX user jcd to Windows user johndoe. A blank username, signified by “”, results in denial of access to the specified Windows user. The mapping is done on a first match basis, so the wildcard entry shown in the last line above is sometimes used to map everyone who doesn’t match the previous entries to some UNIX account such as nobody or “”. This wildcard entry overrides the default username-to-username mapping, however, so using it means that entries for all legal users must be made in the *usermap* file.

The *usermap.cfg* file also supports the username mapping from a single or multiple hosts. The syntax is:

```
[NetAddress:][Domainname\]Windows_username  [direction_sym] [NetAddress:]UNIX_username
```

where

`NetAddress` is an IP address and optionally a subnet specification, or the wildcard *:

192.9.100.14	An IP address
alf.foo.bar.org	A name that resolves to an IP address
192.9.100.0/24	The class C network at 192.9.100
alf.foo.bar.org/255.255.255.0	The class C network at foo.bar.org

`Domainname` is the name of a Windows domain, or *

`Windows_username` is any legal Windows username, the wildcard *, or “”. Names with embedded blanks must be surrounded by double quotes.

`direction_sym` is == (bidirectional), <= or =>. If it is omitted, the default is ==, unless exactly one of the usernames contains a wildcard, in which case the mapping is from the fully specified name onto the wildcard, but not vice versa.

`UNIX_username` is any legal UNIX username, the wildcard *, or “”

Examples of some new legal entries are as follows:

Administrator <= root	Map the UNIX root user to the Windows Administrator account, but don’t map Administrator users to root.
eng\”Tom S” => guest	Map Tom S from the eng domain to the UNIX guest account.
dufus* nobody	Map everyone from the dufus domain to the UNIX nobody account.
reallydumb* => “”	Deny UNIX access to everyone from the reallydumb domain.
“UNIX Users”	Map all UNIX accounts to the Windows account

*	named UNIX User.
corp* *	Map all UNIX users to the same name in the corp domain, and vice versa.
*\root => nobody	Map all requests from a Windows user named root to the UNIX nobody account. This is useful to override the default—and unsafe—mapping root == root.
guest <= administrator	Map all requests from a UNIX user named administrator to the Windows guest account, again for security reasons.
** => “”	Deny access to all Windows users not mentioned on a previous line in the *usermap.cfg* file.
“” <= *	Deny access to all UNIX users not mentioned on a previous line in the file.
eng* <= pizzabox:*	Map all requests from the pizzabox UNIX system to the accounts of the same name in the eng domain.

The first entry above, Administrator <= root, maps root to Administrator, thereby effectively allowing anyone with a Linux box and network access to the filer to access Windows files in mixed and NTFS qtrees with full administrative privilege. To help cut down on the obvious security concerns this raises, mapping can be restricted to operate only on certain IP addresses or subnets. In the examples below, /n means “use the first n bits of this address.” For a class C network, n is 24; for a class B network, it is 16; and for a class A network, it is 8. Standard IP address/netmask syntax may also be used, as in the example in the syntax section.

“”<= 192.10.100.0/24	Deny all NFS requests from the 192.10.100 subdomain.

192.10.100/24:eng* => guest	Map all requests from the 192.10.100 subnet whose users are members of the eng domain to the UNIX guest account
\* == pizzabox/24:	Map all requests from the class C subnet pizzabox is on to the corresponding name on the domain the filer is registered with, and map users from all domains to the corresponding UNIX account. This may result in multiple users from different domains getting mapped to the same UNIX account. The NetAddress qualifier is used for matching UNIX users only when mapping to Windows, not when matching Windows users matching to UNIX.

For more information on the usermap file, see the *System Administrators' Guide.* It should be obvious that usermap entries must be chosen with care, and that thought must be given to defensive mappings that minimize the security risk posed by mappings that allow UNIX users to access Windows files with more permissions than is desired and vice versa. These defensive mappings can also cause problems, however. The ones above that deny access to all users not previously mentioned in the file, for example, can lock all multiprotocol users out of the filer if they are placed at the top of the *usermap.cfg* file.

The *usermap.cfg* file also permits comments. Anything on a line coming after the character # is ignored. The default *usermap.cfg* file that ships with the filer contains some commented defensive entries to assist in creating the file that will actually be used for your installation.

6.4 Authentication

- In an NFS context, authentication is done by the client not the server. CIFS authentication is the server's responsibility. Several modes of CIFS authentication are supported by NetApp filers:
  - Windows domain authentication is supported by NetApp filers. (NetApp filers cannot act as Windows domain controllers; Domain authentication is implemented as "pass-through" authentication. NetApp filers join the domain as member machines via NetLogon.)
  - Windows workgroup authentication. Local users are created on the filer, and the security database is local to the filer.

  - o Plaintext password authentication is a supported alternative to domain authentication. The filer's */etc/passwd* file or, if NIS is used on the filer, the NIS *passwd* map can be used for authentication.

NFS provides some limited granularity at the file system or mount point level—i.e., which machines can access an NFS mount point with what global permissions (e.g., read only), as per the */etc/exports* file. For example, a list of specific NFS clients or groups of clients (as specified in the */etc/netgroup* file) might have full read-write access, whereas another list of clients might only have read-only privileges. To simplify administration of multiple machines in larger UNIX/NFS environments, NetApp filers can participate in NIS (Network Information System, formerly known as "yp" for "yellow pages") as an NIS client. NIS allows a system administrator to maintain centrally located map files with information typically used for authentication such as the */etc/passwd* file.

In a Windows networking environment, the client is not implicitly trusted. Therefore the server must authenticate all accesses. This can still be done by checking for a user entry in the filer's local */etc/passwd* file, and that file can still be maintained via NIS (useful in multiprotocol environments, though irrelevant to CIFS-only sites). The primary weakness of this approach is that users' passwords transit the network in plain text (unencrypted). Furthermore, most CIFS sites, whether heterogeneous or CIFS-only, find it more convenient and effective to authenticate CIFS users by means of a Windows domain controller.

The domain controller (DC) stores users' credentials and can validate all authentication requests on behalf of other machines (e.g., NetApp filers) in the domain. This is known as "pass-through" authentication.

By joining the domain and generating their own challenges, NetApp filers are able to:

- Use a secure channel for validating users' credentials (including passwords) with the DC
- Eliminate the possibility of "man in the middle" security attacks
- Accelerate completion of authentication
- Obtain needed information about each user (e.g., group affiliation)

Domain authentication is recommended by NetApp for its stronger security and greater simplicity of administration.

6.5 Locking

- Some application software requires that a file (or a range of bytes within a file) be "locked" for exclusive read or write access.

- Locking is implemented differently for NFS and CIFS. Locking in the NFS context is advisory, *not* enforced, and typically not used very much. However, for PCs using CIFS locking is mandatory, *strictly* enforced, and in many cases required by the application software.

- NetApp filers support both methods of file locking.

Locks can be granted for file access such that an application running on a particular client can have exclusive use of a file for writing or reading. Locks may apply on a whole-file basis (requested when the file is opened) or for a range of bytes within a file (requested after the file is already open). Under most versions of UNIX, applications have no method for requesting a whole-file lock except to specify a byte range spanning the entire length of the file (after the file has been opened). However, an NFS server's Lock Manager usually does understand whole-file lock requests that can originate on non-UNIX NFS clients.

NFS clients do not actually "open" a file on the server, so appropriately respectful behavior is voluntary, requiring that the client check (usually by requesting a lock from the Lock Manager for themselves) whether a lock exists for a file. In other words, in a UNIX/NFS context, applications must police themselves with respect to locks. This type of locking is known as "advisory locking" and is an admittedly weak model[17].

NFS itself does not implement locks. This is done outside NFS by a side protocol usually referred to as the Lock Manager, or more commonly as the "lock daemon" or "lockd" (on most UNIX-based NFS servers, `rpc.lockd` is the lock daemon). Unfortunately, the Lock Daemon has historically had many implementation bugs, and this has discouraged developers from writing their applications to check for the existence of locks on files. This means that an application running under UNIX on an NFS client will typically access a file without checking for the existence of another client's lock or asking for a lock itself.

With CIFS, the situation is very different. A CIFS client expects its locks to be enforced by the server. Applications are routinely written to use locks and to rely on their enforcement. A CIFS client can also request an *oplock* (opportunistic lock) for a file, such that the file server allows the client to locally cache the file and not write any file updates back to the server until the oplock is relinquished or the file is closed. If the server receives a read request for the file from another client, the server will retract and reissue the lock as a "batch" oplock, allowing both clients to cache reads but not write. If any client wishes to write, the server reinstates the ordinary mechanisms of read and write lock enforcement.

[17] In NFSv4, both advisory locking and mandatory locking are supported, the addition of mandatory locking would greatly improve NFS performance.

Emulated CIFS file servers like SAMBA cannot truly enforce respect for CIFS locks among NFS clients. The emulated CIFS file service runs as an application, outside of the kernel and independent of the underlying file system, potentially allowing NFS accesses to "go around" its locks. To mitigate the likelihood of such events, SAMBA (or a commercial CIFS-emulating equivalent) registers with the lock daemon all PC locks it has granted. However, most UNIX applications have not been written to query the historically unreliable lock daemon before opening files; therefore, NFS violations of PC locks are routine on such CIFS-emulating servers.

Before a NetApp filer responds to any file access request, it first verifies that the client has the appropriate permissions. For a CIFS file access request, the filer also checks—in the WAFL file system—that there are no outstanding locks that would exclude the attempted access. For an NFS file access request, this same lock check is made, *plus* the application software running on the NFS client may in some cases interact with the Lock Manager. PC clients will not interact with the Lock Manager unless they are using (PC) NFS instead of CIFS.

6.6 Character Sets

WAFL fully supports UNICODE, a 2-byte encoding system that provides the capacity to encode all of the characters used for the written languages of the world. Windows is based on UNICODE and can theoretically support (correctly write, view, and access) file names with any of the supported characters at the same time. The fonts included with the operating system, however, do not contain all of the UNICODE characters. Windows 2000 and third-party fonts for Windows support most, if not all, of the UNICODE characters.

UNIX/NFS does not support UNICODE. Instead it supports simple code pages that are subsets of locales. On a given network, all UNIX machines and filers should be set to the same locale so that file names will be displayed uniformly. Since UNIX/NFS is case sensitive, the operating system does not need to interpret the bit string representing the file name. This is contrary to the operation of NTFS, which needs to interpret case for the entire UNICODE character range.

Windows 9x systems support a subset of UNICODE for file names only. User names and share names are encoded using code pages. These systems should therefore have the language set to the same as that of the file server.

Windows NT, 2000, 2003, and XP use UNICODE for Network Neighborhood displays. For simplicity and better manageability (from, say, a centralized support location) the same language should be used across the entire Windows network. An even simpler solution would be to standardize on ASCII-based machine and domain names.

On a filer, language is set per volume. This is a very useful feature for multinational companies employing UNIX/NFS clients in different countries (locales).

For each file, WAFL keeps a long file name (up to 255 characters) and a DOS-style 8.3 filename. If a CIFS client creates a UNICODE file name containing non-ASCII characters, WAFL will create an ASCII file name so that UNIX/NFS clients can always access the file. In the case where a UNIX/NFS client writes a file containing characters that are illegal for PC clients, an 8.3 file name will be generated. The same is true if there is a case collision. UNIX/NFS generated file names that don't translate to UNICODE are preserved in such a way as to maintain the original bit stream. Data ONTAP supports a variety of languages, simply use the Data ONTAP command "`vol lang`" to retrieve a most current list of supported languages.

6.7 Snapshot Copies

The NetApp WAFL file system has an interesting feature not found in other file systems, called Snapshot copies.

A Snapshot copy is a "frozen" (read-only) point-in-time copy of the file system created by preserving all the pointers to all the disk blocks currently in use at the time of the copy. After a Snapshot copy has been made, changes to files are reflected in updates to the current set of pointers[18], no differently than if no Snapshot copies existed. The disk usage overhead of Snapshot copies is minimized by preserving individual 4KB disk blocks instead of whole files. Snapshot copies can be scheduled to occur automatically on a recurring basis. Data recovery from online Snapshot copies is a much more efficient way compared to tape storage. There can be up to 255 Snapshot copies per volume at any one time.

Files in a Snapshot copy of the file system have the same access privileges as in the active file system. A user with permission to read a file in the active file system will still be able to read it in a Snapshot copy. Users without such permission will still be unable to read the file. Write access privileges simply do not apply—files within a Snapshot copy cannot be changed because Snapshot copies are read-only. Updates to files must be performed on the file's image in the current, active file system.

Backups can be performed via Snapshot copies, allowing users to continue to write to the live, actual file system without endangering the restorability of the backup in progress. In fact, the NetApp `dump` command will automatically create a temporary Snapshot copy for its own use, unless directed otherwise by the system administrator.

Snapshot copies are whole parallel file systems, from the top of the directory tree on down. There is an entry into this parallel directory tree accessible from every directory—a subdirectory named *.snapshot* for UNIX/NFS (and PC software that can handle long file names) within every directory in the active file system. These *.snapshot*

[18] The WAFL file system never overwrites existing blocks in a file. Instead, WAFL always writes new blocks, then updates the pointers in the active file system, and releases the superseded blocks for other use—unless those older blocks are still part of a Snapshot copy.

subdirectories are *invisible* (see below). Within the *.snapshot* subdirectories are lower subdirectories named for the interval in which that particular Snapshot copy was made.

Also, at the top of the directory tree, the Snapshot copy may be entered through a *visible* (see below) directory named *.snapshot* (for UNIX/NFS), *~snapshot* for long-file-name-compatible PC/CIFS clients, and *~SNAPSHT* for PC/CIFS clients that require 8.3 file names.

In order to avoid confusion, except for the top of the directory tree, Snapshot copy entry point subdirectories are not displayed—are *invisible*—in ordinary listings of directory contents, and cannot be revealed by other methods of inspecting a directory's contents. This is accomplished slightly differently in NFS than in CIFS. Aside from simplifying the user's interface with the file system, concealing Snapshot copies also protects against problems such as the UNIX recursive remove (`rm -rf`) failing because of the read-only nature of the contents of a Snapshot copy.

The example below shows the appearance of the Snapshot copy parallel directory tree from the UNIX/NFS perspective.

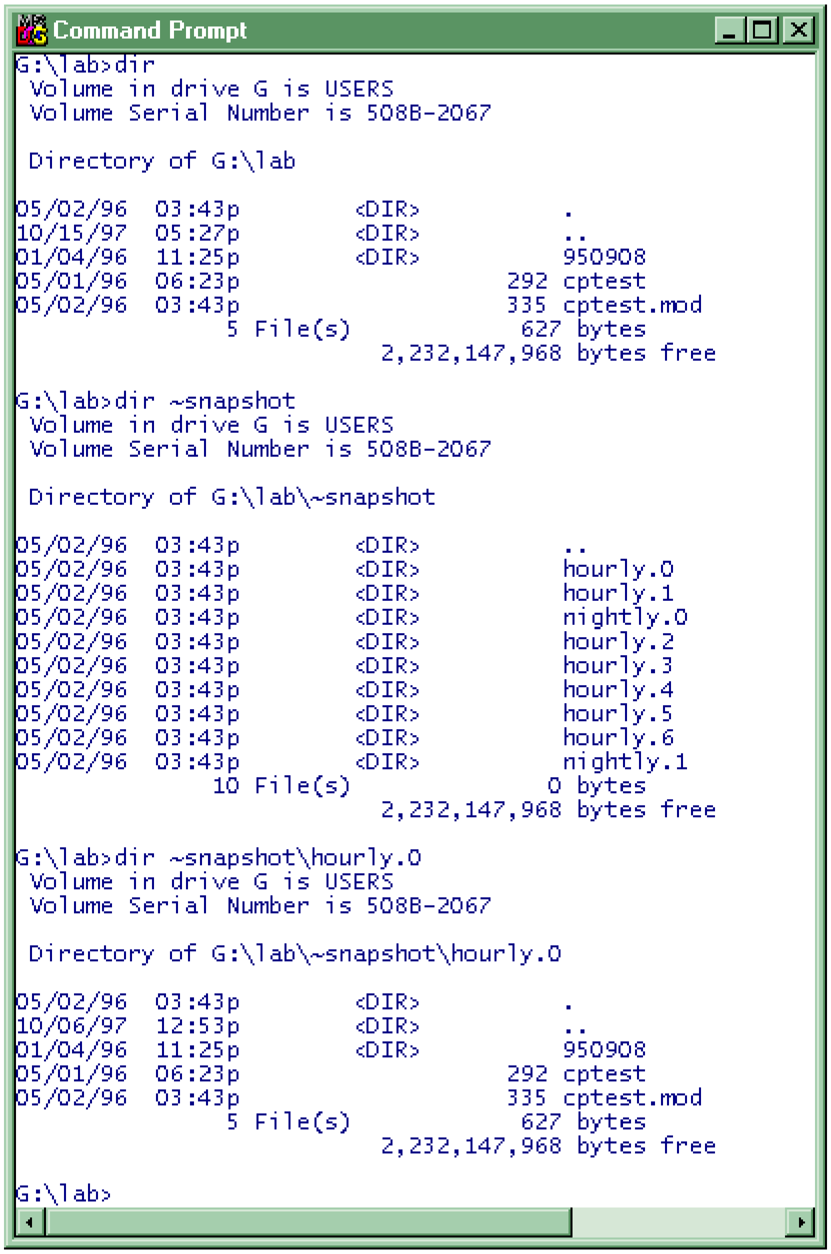
```
# ls -alg ~agy/test
total 444
drwxr-xr-x  2 agy   eng     4096 Nov 18 01:10 ./
drwxr-xr-x 30 agy   eng     8192 Nov 26 19:25 ../
-rw-r--r--  1 agy   eng   197648 Nov 18 01:09 results.1
-rw-r--r--  1 agy   eng   233224 Nov 18 01:09 results.2

# ls -alg ~agy/test/.snapshot
total 48
drwxrwxrwx  2 root  wheel   4096 Nov 27 00:01 ./
drwxr-xr-x  2 agy   eng     4096 Nov 18 01:10 ../
drwxr-xr-x  2 agy   eng     4096 Nov 18 01:10 hourly.0/
drwxr-xr-x  2 agy   eng     4096 Nov 18 01:10 hourly.1/

# ls -alg ~agy/test/.snapshot/hourly.0
total 444
drwxr-xr-x  2 agy   eng       4096 Nov 18 01:10 ./
drwxr-xr-x 30 agy   eng       8192 Nov 26 19:25 ../
-rw-r--r--  1 agy   eng   197648 Nov 18 01:09 results.1
-rw-r--r--  1 agy   eng   233224 Nov 18 01:09 results.2
```

By convention, the `UNIX /bin/ls` command (which lists a directory's contents) suppresses display of file names beginning with the dot character (e.g., the file named *.cshrc*). If the `/bin/ls` command is invoked with the `-a` command line option, the resulting directory listing then displays such "hidden" files. However, even with the `-a` option, display of the file named *.snapshot* will still be suppressed. Only when *.snapshot* is provided as an explicit command line argument will `/bin/ls` "see" it (or its contents). The effect of this can be seen in above example.

PCs have a similar convention, in which the Windows GUI allows users to choose to have "hidden" files suppressed from display. However, unlike the UNIX `/bin/ls` command, the Windows GUI *will* include *~snapshot* (or *~SNAPSHT*) in listings of files together with other hidden files. In other words, although with NFS *.snapshot* is a special category of hidden file, via CIFS the display of *~snapshot* is no more suppressed than other hidden files. Figure 10 shows how Snapshot copies are displayed at the Windows command prompt and Figure 11 shows how they are displayed in the GUI.



```
G:\lab>dir
 Volume in drive G is USERS
 Volume Serial Number is 508B-2067

 Directory of G:\lab

05/02/96  03:43p        <DIR>          .
10/15/97  05:27p        <DIR>          ..
01/04/96  11:25p        <DIR>          950908
05/01/96  06:23p                   292 cptest
05/02/96  03:43p                   335 cptest.mod
               5 File(s)            627 bytes
                          2,232,147,968 bytes free

G:\lab>dir ~snapshot
 Volume in drive G is USERS
 Volume Serial Number is 508B-2067

 Directory of G:\lab\~snapshot

05/02/96  03:43p        <DIR>          ..
05/02/96  03:43p        <DIR>          hourly.0
05/02/96  03:43p        <DIR>          hourly.1
05/02/96  03:43p        <DIR>          nightly.0
05/02/96  03:43p        <DIR>          hourly.2
05/02/96  03:43p        <DIR>          hourly.3
05/02/96  03:43p        <DIR>          hourly.4
05/02/96  03:43p        <DIR>          hourly.5
05/02/96  03:43p        <DIR>          hourly.6
05/02/96  03:43p        <DIR>          nightly.1
              10 File(s)              0 bytes
                          2,232,147,968 bytes free

G:\lab>dir ~snapshot\hourly.0
 Volume in drive G is USERS
 Volume Serial Number is 508B-2067

 Directory of G:\lab\~snapshot\hourly.0

05/02/96  03:43p        <DIR>          .
10/06/97  12:53p        <DIR>          ..
01/04/96  11:25p        <DIR>          950908
05/01/96  06:23p                   292 cptest
05/02/96  03:43p                   335 cptest.mod
               5 File(s)            627 bytes
                          2,232,147,968 bytes free

G:\lab>
```

Figure 10) Snapshot copies in Windows command line prompt.

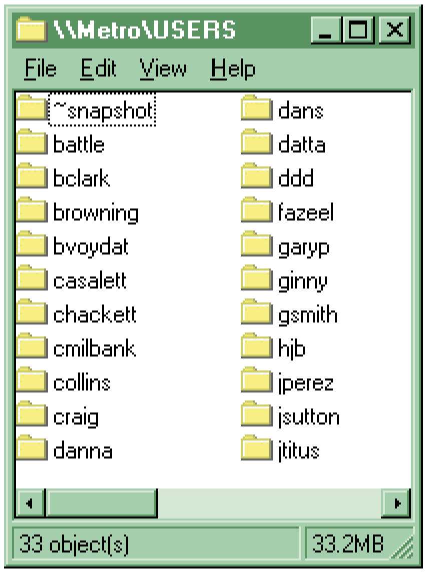
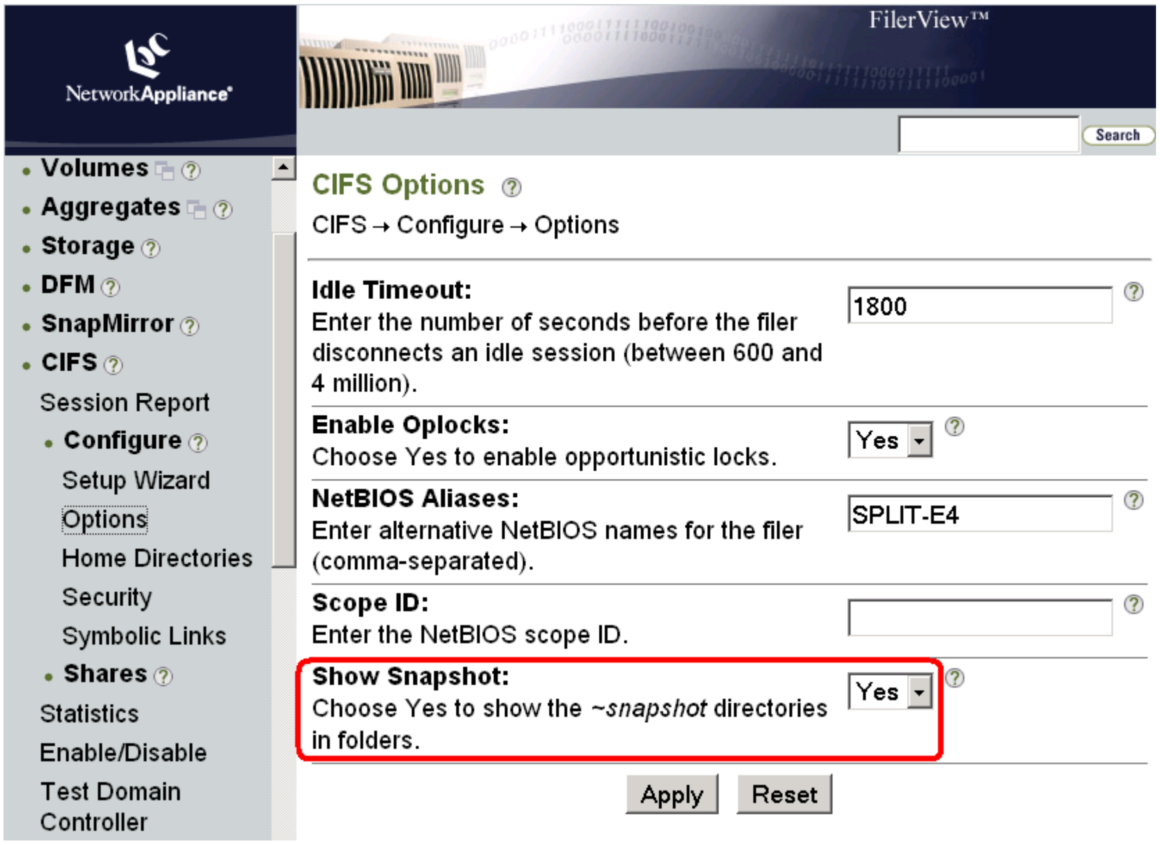


Figure 11) Snapshot copy folder in Windows.

The ~*snapshot* directory can be completely hidden (even to Windows clients that are configured to “view hidden files”), yet remain browseable by explicitly specifying the directory name. An easy way to configure this is via FilerView (see Figure 12).



CIFS Options

CIFS → Configure → Options

Idle Timeout: Enter the number of seconds before the filer disconnects an idle session (between 600 and 4 million). 1800

Enable Oplocks: Choose Yes to enable opportunistic locks. Yes

NetBIOS Aliases: Enter alternative NetBIOS names for the filer (comma-separated). SPLIT-E4

Scope ID: Enter the NetBIOS scope ID.

Show Snapshot: Choose Yes to show the ~*snapshot* directories in folders. Yes

Figure 12) Configuring Snapshot visibility through FilerView

Snapshot copies provide users with a method to recover almost instantly from accidental file deletions or to return to older versions of files. This can lift a significant burden from the shoulders of the system administration staff, in that fewer recoveries of files from backup tapes are likely to be requested by users. Consequently, NetApp customers often cite Snapshot copies as a resource they could not appreciate fully until they had experience with them, but that they came to increasingly depend upon over time. Furthermore, Data ONTAP Snapshot copies can also be reverted to become the active file system or propagated to remote machines for mirroring of data.

For more details about Snapshot copies, see Section 2 of NetApp Technical Report 3002 [TR-3002].

6.8 Filer Management with DataFabric® Manager

As a customer's need for storage grows, so does the number of filers. Although Data ONTAP software aims at reducing the complexity of storage management by providing features such as Snapshot and multiprotocol access, system administrators still encounter pain points in these areas as the company grows:

- Too many isolated “storage islands” and too many point solutions to manage these storage islands
- Complex processes for allocating and expanding storage
- Need for visibility into where each device is and how they are being used
- Need for 24x7 access; need to know that something is going wrong before the failure occurs; extremely expensive downtimes
- Need for unified view of where the bottlenecks are (CPU, network, or the application?)
- Extremely complex management in mixed NAS and SAN environment; UNIX and Windows environments (need to make sure IT can use data across platforms/OS)
- Need for visibility into who is using what, resource sharing, what filers are overloaded and what filers are underutilized
- Difficult advance planning (knowing when to add capacity)

NetApp DataFabric Manager (DFM) is a powerful, easy-to-use application for managing a NetApp storage/NetCache infrastructure from a single console. DFM addresses these pain points by allowing an organization to rapidly deploy, provision, and manage a complete enterprise storage network. For more information on DFM, refer to TR3296.

7. Conclusions

Multiprotocol filing facilitates easier sharing of data in heterogeneous environments without compromising security. This drives cost savings, streamlined administration, and enhanced productivity. As mixed environments become increasingly common, the need for more multiprotocol filing will also increase.

The unique elements of Data ONTAP software—the specialized microkernel, the WAFL file system, support for NDMP, fully integrated and dynamically expandable RAID-protected storage, etc.—make native multiprotocol filing a practical reality for environments employing NFS, CIFS, and HTTP. The WAFL file system’s unique Snapshot capability is especially useful for making fully restorable backups with minimal (if any) impact on users’ access to their data. NetApp has also supplied a suite of essential features and utilities that make deployment in a multiprotocol environment easier, including tools for dynamically mapping Windows user accounts to UNIX UIDs, or assigning all Windows users the same UNIX user account, migrating Windows domain user account information to UNIX password and group databases, and so on. Foreign language support is provided by a unique dynamic mapping of ASCII (for NFS) and UNICODE (for Windows).

Compared to alternatives like SAMBA or other emulated CIFS file service applications, the NetApp multiprotocol filer's native implementation provides stronger support for the CIFS security model and file locking.

8. References

[MS-96c] Microsoft's CIFS protocol specification reference

[NCD-96] NCD offers a multiple-user Windows product called WinCenter

[Tek-96] Tektronix offers a multiple-user Windows product called WinDD

[TR-3001] Dave Hitz, A Storage Networking Appliance

[TR-3002] Dave Hitz, James Lau, and Michael Malcolm, "File System Design for an NFS File Server Appliance." *Winter USENIX Conference Proceedings*, USENIX Association, Berkeley, CA, January, 1994. This paper is also available directly from NetApp as Technical Report 3002

[TR-3085] "The NFS Version 4 Protocol," TR-3085, Network Appliance

[TR-3362] "How Network Appliance Devices Join Active Directory," TR-3362, Network Appliance

[TR-3367] "NetApp Filer in a Microsoft Windows Environment," TR-3367, Network Appliance