HITACHI
Inspire the Next

Hitachi Data Systems SU 12.x

Network File System (NFS) Version 4 Feature Description

Hitachi Data Systems

MK-92HNAS056-00

Hitachi Data Systems products and services can be ordered only under the terms and conditions of Hitachi Data Systems' applicable agreements. The use of Hitachi Data Systems products is governed by the terms of your agreements with Hitachi Data Systems.

Hitachi is a registered trademark of Hitachi, Ltd., in the United States and other countries. Hitachi Data Systems is a registered trademark and service mark of Hitachi, Ltd., in the United States and other countries.

Archivas, Dynamic Provisioning, Essential NAS Platform, HiCommand, Hi-Track, ShadowImage, Tagmaserve, Tagmasoft, Tagmasolve, Tagmastore, TrueCopy, Universal Star Network, and Universal Storage Platform are registered trademarks of Hitachi Data Systems Corporation.

AIX, AS/400, DB2, Domino, DS8000, Enterprise Storage Server, ESCON, FICON, FlashCopy, IBM, Lotus, OS/390, RS6000, S/390, System z9, System z10, Tivoli, VM/ESA, z/OS, z9, zSeries, z/VM, z/VSE are registered trademarks and DS6000, MVS, and z10 are trademarks of International Business Machines Corporation.

All other trademarks, service marks, and company names in this document or website are properties of their respective owners.

Microsoft product screen shots are reprinted with permission from Microsoft Corporation.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Some parts of ADC use open source code from Network Appliance, Inc. and Traakan, Inc.

Part of the software embedded in this product is gSOAP software. Portions created by gSOAP are copyright 2001-2009 Robert A. Van Engelen, Genivia Inc. All rights reserved. The software in this product was in part provided by Genivia Inc. and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

The product described in this guide may be protected by one or more U.S. patents, foreign patents, or pending applications.

Notice of Export Controls

Export of technical data contained in this document may require an export license from the United States government and/or the government of Japan. Contact the Hitachi Data Systems Legal Department for any export compliance questions.

Document Revision Level

Revision	Date	Description
Version 0	10/29/14	Document created by GSS FCS Services Engineering

Contact

Hitachi Data Systems
2845 Lafayette Street
Santa Clara, California 95050-2627
https://portal.hds.com

North America: 1-800-446-0744

Table of Contents

Overview

Contacting Hitachi Data Systems Support

The Hitachi Data Systems customer support staff is available 24 hours a day, seven days a week. If you need technical support, log on to the Hitachi Data Systems Portal for contact information: https://portal.hds.com.

Kerberos Authentication

With NFS v2, NFS v3, and NFS v4, the RPCSEC_GSS security style (defined in RFC 2203) provides authentication, integrity, and privacy, which uses GSS-API to interface to specific security mechanisms that provide these services.

Technology Overview

Kerberos provides a mechanism for entities (*principals*) to authenticate to each other and securely exchange a session key that can be used to sign and seal messages during a particular association.

RPCSEC_GSS multiplexes many security mechanisms, signing and sealing algorithms, and security services (authentication, integrity, and privacy) over a single RPC security flavor. For NFS v3, specific combinations are mapped to *pseudo* RPC security flavors, so that they can be communicated to the client using the mount protocol. NFS v4 provides a SEC_INFO call that can be used to enumerate supported combinations (at any point in the file system tree). NFS defines one signing and one sealing algorithm that must be supported when using Kerberos, DES MAC MD5, and 56-bit DES, respectively.

For more information, see:

- RFC 1510 (Kerberos)
- RFC 1964 (Kerberos GSS-API mechanism)
- RFC 2743 (GSS-API)
- RFC 2203 (RPCSEC_GSS)
- RFC 2623 (NFS v2/v3 use of RPCSEC_GSS/Kerberos)

Configuration Options

Secure NFS requires configuration of the NFS server's Kerberos principal name and secret keys. Kerberos-related configuration is setup both globally and on a per-EVS basis. The NFS host name is configured on a per-EVS basis.

Export configuration defines which security types can be used to access the export, which can include one or more of the following:

- UNIX
- Kerberos with authentication only
- Kerberos with integrity
- Kerberos with privacy

Performance is likely to suffer if integrity or privacy is enabled.

Microsoft KDC Support

The HNAS NFS system supports Kerberos authentication with active directory as a key distribution center (KDC). To configure Kerberos-enabled NFS, it is necessary to export the *keytab* file from the KDC. Manually import it into HNAS, set the realm, and the HNAS does the rest.

Protocol Characteristics

This section includes detailed information about the HDS SU 12.x NFS v4 implementation.

Pseudo File System

NFS v4 clients do not connect directly to NFS exports as in versions 2 and 3. Instead, all clients connect to the root of the pseudo file system, which is a virtual directory structure that the clients can traverse to reach the exports. The pseudo file system is generated automatically from the NFS exports, meaning that no extra configuration is necessary when upgrading from version 2 or 3 to version 4. NFS export access options (for example: ***(ro)**) are enforced over NFS v4 as they are over other versions of the protocol.

There are two NFS procedures that allow a client to get an initial file handle. They are PUTROOTFH (put root file handle) and PUTPUBFH (put public file handle). PUTROOTFH sets the Current File Handle to the handle of the root of the pseudo file system. PUTPUBFH sets the Current File Handle to the public file handle. Each EVS has a separate public file handle, as determined by the administrator, which can identify any directory in any of the EVS file systems, either pseudo or real. For example, if an administrator has a set of public data (for instance, data on a corporate intranet), they can set the public file handle to be the root of that public data.

See RFC 3530 for more information about the pseudo file system and its uses.

Transport

RFC 3530 states that:

> *Where an NFS version 4 implementation supports operation over the IP network protocol, the supported transports between NFS and IP MUST be among the IETF- approved congestion control transport protocols, which include TCP and SCTP. To enhance the possibilities for interoperability, an NFS version 4 implementation MUST support operation over the TCP transport protocol.*

For this reason, HNAS supports NFS v4 over TCP (for example, UDP is not a supported transport). This is different from protocol versions 2 and 3, which operate over both TCP and UDP.

The main incentive for adding this restriction is that the duplicate request cache used by versions 2 and 3 has had a number of problems that have caused protocol errors to occur. The fact that NFS v4/5 are stateful makes it much more complicated than versions 2 and 3, and trying to fit a reliable duplicate request cache into the protocol will be error-prone and non-trivial. Supporting only TCP means that the transport layer will be reliable, removing the need for an NFS v2/3-style duplicate request cache altogether. RFC 3530 confirms this by stating:

> *When processing a request received over a reliable transport such as TCP, the NFS version 4 server must not silently drop the request, except if the transport connection has been*

broken. Given such a contract between NFS version 4 clients and servers, clients must not retry a request unless one or both of the following are true:

- The transport connection has been broken
- The procedure being retried is the NULL procedure

Note that it is still necessary to track of the last response that was sent on a connection, to cope with connection reestablishments.

File Handles

NFS v4 file handles on HNAS are persistent, meaning that they will not change over server reboots. Providing persistent file handles has a number of advantages, mainly associated with crash recovery (see RFC 3530 for more details). Furthermore, HNAS file handles will persist over file system migration, as a file handle contains no EVS-specific information. RFC 3530 states that:

> *...in the case that two different path names when traversed at the server terminate at the same filesystem object, the server SHOULD return the same file handle for each path.*

The HNAS NFS v4 implementation adheres to this recommendation only if the two paths transition from the pseudo file system to the real file system at the same point. To clarify, consider the following pseudo file system links:

Pseudo File System		***Real File System***
/home/root	➲	/
/home/fred	➲	/users/fred
/home/joe	➲	/users/joe

Also, imagine two hard links: */users/fred/myfile* and */users/joe/myfile*. These two hard links resolve to the same underlying file, so they have the same file handle within the real file system. Traversing the following two paths results in the same NFS v4 file handle:

/home/root/users/fred/myfile

/home/root/users/joe/myfile

This is because they both transition from the pseudo file system to the real file system at */home/root*. Conversely, traversing the following two paths results in different NFS v4 file handles, as they transition from the pseudo file system to the real file system at different points:

/home/fred/myfile

/home/joe/myfile

Although this violates the recommendation in RFC 3530, as stated above, it is unlikely to cause any functional problems. It is likely that the client's cache will not be as efficient as it might otherwise be, as it has to allocate one cache entry for each *myfile*. However, this slight performance hit should be the extent of the negative effects.

Open State

Note the following:

- A *clientid* is a server-assigned value that uniquely identifies an NFS v4 client for the duration of its *session* with the server. The *clientid*s generated by HNAS will not conflict with *clientid*s previously allocated by the server, even if either the client or the server has rebooted in the meantime.
- A *stateid* is a server-assigned value that uniquely identifies state, such as an open file or a lock. The *stateid*s generated by HNAS will not conflict with *stateid*s previously allocated by the server, even if either the client or the server has rebooted in the meantime. In addition, HNAS will be able to detect stale *stateid*s (issued by a previous invocation of the server), old *stateid*s (issued by the current invocation of the server, but no longer identifying valid state) and bad *stateid*s (never issued by any invocation of the server).
- A *lock_owner* is a client-assigned value that uniquely identifies the owner of a lock, such as a process on the client. RFC 3530 is not specific on how the *lock_owner* should be treated. Parts of the RFC seem to imply that a given *lock_owner* should maintain session-wide state, whereas other parts imply that state is only maintained on a per-file basis. This distinction is important, as the *sequenceId* member of the *stateId* structure must be incremented every time a given *lock_owner* locks a file. If the client expects the same *sequenceId* to be incremented every time a *lock_owner* locks any file within a session, but the server maintains independent *sequenceIds* per file, the client and server will not be consistent and will not operate correctly.

HNAS maintains separate *lock_owner* state per file, as Spencer Shepler, a contributor to RFC 3530, clarifies in the following:

> *When an OPEN is done for a file and the lock_owner for which the open is being done already has the file open, the result is to upgrade the open file status maintained on the server to include the access and deny bits specified by the new OPEN as well as those for the existing OPEN. The result is that there is one open file, as far as the protocol is concerned, and it includes the union of the access and deny bits for all of the OPEN requests completed. Only a single CLOSE will be done to reset the effects of both OPENs.*

When HNAS receives an OPEN request containing a file handle, it will not know whether it is the first OPEN for the target file, or whether the file is already open. For this reason it must always index into the file table by file handle, to see whether the file is open already, before continuing to process the request. RFC 3530 also states that:

> *When the server chooses to export multiple file handles corresponding to the same file object and returns different file handles on two different OPENs of the same file object, the server MUST NOT "OR" together the access and deny bits and coalesce the two open files. Instead the server must maintain separate OPENs with separate stateids and will require separate CLOSEs to free them.*

Indexing into a session's file table by file handle ensures that if two different file handles resolve to the same file system object, as they can do on HNAS, the OPENs is not combined.

Mixing Security Models

HNAS allows both UNIX and Windows clients to access files and directories simultaneously. So, a file must be able to have both UNIX security (for example: a UID, GID, and mode) and Windows security

(for instance, a security descriptor). Clearly, a file that has been secured from a Windows client, using a security descriptor, will not be accessible from a UNIX client unless the security descriptor allows it. Equally, a file that has been secured from a UNIX client, using the mode, cannot be accessible from a Windows client unless the mode allows it. The rest of this section describes how this is achieved within the HNAS's file system.

Object Security

Every file system object has a UID, GID, and mode. This is true even if a customer only uses CIFS and has never licensed NFS. However, not every object has a security descriptor. So, an object can be in one of two states. Either it only has UNIX security, or it has both UNIX and Windows security.

Security Modes

HNAS supports two security modes: M*ixed mode* and *UNIX mode*. In mixed mode, files are permitted to have security descriptors. In UNIX mode, they are not. In the following sections, mixed mode operation should be assumed unless otherwise stated.

	In a directory with UNIX security only	***In a directory with UNIX security and a security descriptor***
Create over NFS v2/3 (pre SU 6.x)	New file has UNIX security only.	New file has UNIX security only.
Create over NFS v2/3 (SU 6.x)	New file has UNIX security only.	New file has SD; DACL is either set as given in request, or inherited from the parent and modified with mode in request.
Create over NFS v4/5	New file has SD; DACL is either set as given in request, derived from mode in request, or derived from umask.	New file has SD; DACL is either set as given in request, or inherited from the parent and modified with mode in request.
Create over CIFS	New file has SD; DACL is either set as given in request, or derived from umask.	New file has SD; DACL is either set as given in request, or inherited from the parent and derived from umask.

The following table highlights the effect various operations have on a file's security when in mixed mode.

		On a file with UNIX security only	*On a file with UNIX security and a security descriptor*
Security set over NFS v2/3 (pre SU 6.x)	chown/chgrp	UID/GID updated	SD discarded, UID/GID updated
	chmod	Mode updated	SD discarded, mode updated
Security set over NFS v2/3 (SU 6.x)	chown/chgrp	UID/GID updated	Ownership updated, current mode applied to DACL[1]
	chmod	Mode updated	Mode applied to DACL
Security set over NFS v4/5	chown/chgrp	SD derived from UNIX security, ownership updated	Ownership updated, current mode applied to DACL[1]
	chmod	Mode updated	Mode applied to DACL
	Set DACL	SD derived from UNIX security, DACL set as given	DACL set as given
Security set over CIFS	Ownership change	SD derived from UNIX security, DACL frozen[2], ownership updated	DACL frozen[2], ownership updated
	Set DACL	SD derived from UNIX security, DACL set as given	DACL set as given

- *File renamed*: The file's security remains unchanged, irrespective of the type of client that does the rename.
- *Link created*: The file's security remains unchanged, irrespective of the type of client that does the link.

If a file has a security descriptor, it is used to grant or deny access to the file. This will be the case even for users accessing the file over NFS. A security descriptor can hold much more detailed security information than a UNIX mode, which is why it is preferred. If a file does not have a security descriptor, the mode must be used to check access rights instead, even for users accessing the file over CIFS.

User and Group Mappings

If a file has both Windows and UNIX security, it has two owner fields and two group fields. One owner and group pair is in its security descriptor and the other is its UID and GID. Irrespective of whether the file is accessed from a UNIX or a Windows client, the owner and group are reported as the same.

[1] This ensures the new owner/group has the same permissions as the old owner/group, which is the behavior expected by Posix.

[2] SIDs that track ownership (for example, OWNER@/GROUP@) are replaced with the actual owner/group. This ensures the interpretation of the DACL does not change (the behavior expected by CIFS clients).

However, the security descriptor stores SIDs, whereas the UID and GID store 32-bit values. As a result, mapping is required to ensure that the same user and group are reported.

Mappings can be set up between UNIX-to-NFS v4 and Windows-to-NFS v4 users and groups in a manner similar to how mappings are set up between UNIX and Windows users and groups. As with UNIX-to-Windows mappings, mappings to and from NFS v4 users and groups can be either manual or automatic. However, using manual mappings might not just be a case of looking up a user name in a table.

As explained in Object Security, clients in different DNS domains could generate different NFS v4 user names for the same UNIX user. For example, consider the following manual NFS v4-to-UNIX mapping:

- jdoe@hds.com ➲ jdoe, 550

If HNAS receives a CREATE or a SETATTR containing the user name *jdoe@dev.hds.com*, it treats *jdoe@dev.hds.com* and *jdoe@hds.com* as equal, so the manual mapping above is sufficient to allow the HNAS system to map correctly.

Automatic mappings between NFS v4 and UNIX users and groups are done in the same way as they are between Windows and UNIX users and groups. The *@dns_domain* portion of the NFS v4 user name is removed, and the resulting user name (*jdoe* in the example above) is converted to a UNIX ID using NIS, LDAP, or a manual lookup.

Automatic mappings between NFS v4 and Windows users and groups are more interesting. As well as the DOMAIN\user format that is traditionally used to represent Windows users, *user@dns_domain* is also a valid representation in ADS environments. It is possible that a user's NFS v4 and Windows user names can be identical. This being the case, HNAS sends the NFS v4 user name to the domain controller as is, in order to obtain an equivalent Windows SID for the user.

When HNAS maps an NFS v4 user to a Windows user automatically, it first examines the *@dns_domain* portion of the NFS v4 user name. If the *@dns_domain* portion matches HNAS's fully-qualified Windows domain name, a Name2SID lookup is performed on the domain controller, using the NFS v4 user name as is.

However, it is possible that the NFS v4 user's domain and HNAS's Windows domain belong to the same parent domain, but do not match exactly. For example, the NFS v4 user name could be *jdoe@dev.hds.com* and HNAS's fully-qualified Windows domain name could be *protocols.terastack.hds.com*. In this case, HNAS first tries to lookup the name *jdoe@dev.hds.com* on the domain controller, in case there is a trust relationship set up between *dev.hds.com* and *protocols.terastack.hds.com*. If this lookup fails, HNAS replaces the *@dns_domain* portion of the NFS v4 user name with the Windows domain name before doing the Name2SID lookup on the domain controller. Using the above example, it would lookup one of the following names:

- *jdoe@protocols.terastack.hds.com*
- PROTOCOLS\jdoe

It is also possible that the NFS v4 user's domain and HNAS's Windows domain do not belong to the same parent domain. For example, the NFS v4 user name could be *jdoe@synaxia.com* and HNAS's fully-qualified Windows domain name could be *protocols.terastack.hds.com*. In this case, HNAS tries to lookup the name *jdoe@synaxia.com* on the domain controller, in case there is a trust relationship set up between *synaxia.com* and *protocols.terastack.hds.com*.

Mapping Failures

A user or group mapping will not always succeed. A mapping might fail because:

- HNAS fails to get the Windows user name from the domain controller.
- HNAS fails to get the UID from the NIS server.
- A manual mapping is not configured for the current user and automatic mappings are disabled.
- Automatic mappings are enabled but fail.

In these cases, HNAS will not be able to perform the mapping, so it will not be able to keep the file's owner and group consistent between security models. Types of mapping failures include:

- Windows/NFS v4-to-UNIX
- Windows-to-NFS v4
- NFS v4-to-Windows

In the case of Windows/NFS v4-to-UNIX user and group mappings, note that the HNAS system only has to perform user and group mappings for files that have security descriptors. Furthermore, a file will only have a security descriptor if either it has been created or its security has been set over CIFS or NFS v4. As a result, when creating or modifying a security descriptor, HNAS always has a Windows or NFS v4 *access token* available, so it is always able to set the Windows/NFS v4 owner and group correctly. This means that a user and group mapping failure is effectively a failure to set the UID or GID.

If a user mapping fails, the UID is set to 65534. Similarly, if a group mapping fails, the GID is set to 65534. The file will effectively not be owned by anybody in the UNIX security model. In addition to setting the UID and GID, one or both of the following flags are set in the file's metadata:

- 0x01 – UIDMappingFailure
- 0x02 - GIDMappingFailure

A file with a failed UID or GID mapping is not permanent. Instead, its mappings are retried automatically the next time it is accessed after any of the following events occurs:

- The manual user or group mappings modifies
- The automatic mappings setting changes
- A 15-minute *retry mappings* timer starts

To enable the HNAS to detect such an event, each file system has a *security mappings version number*, stored in its *dynamic volume config block*. This is an incrementing counter that is initialized to 0 when the file system is first formatted. Each time a file is created within that file system, the file system's current security mappings version number is stored in the file's metadata.

When one of the events occurs, the security mappings version number is incremented on every mounted file system. The next time a file is accessed on a given file system, HNAS can detect whether the security mappings have changed since the file's UID and GID were mapped by comparing the security mappings version number stored in the file's metadata with the file system's current value. To ensure failed security mappings are retried on unmounted file systems, a file

system's security mappings version number is also incremented every time the file system is mounted.

As previously stated, a file's UID and GID mappings are retried the next time it is accessed after one of the aforementioned events occurs. The next time an operation is targeted at the file, HNAS repeats the user and group mapping process. If the mapping still fails, the latest security mappings version number is stored in the file's metadata to ensure the mappings are not retried unnecessarily, and the operation continues. However, if the mapping now succeeds, one of two things will occur. First, the HNAS attempts to write the correctly mapped UID and GID to disk. If this succeeds, all is well and the operation continues as normal. However, if this fails (possibly because the file system is mounted read-only, is full, or the file is in a snapshot) the correctly mapped IDs are not writable to disk. They are stored with the file in cache.

While the file remains in cache, the correctly mapped UNIX IDs are returned along with the file's attributes, so that all appears normal from the client's point of view. However, after the file has gone out of cache, the mappings must be reacquired the next time it is accessed. This can make it reasonably expensive to continually access files with mapping failures in snapshots.

The state of a file's security mappings can be examined by using the security command on the FSM. Lines such as the following are displayed below the file's security descriptor and UNIX security:

```
UID is not mapped correctly
GID is mapped correctly.
Mappings will not be retried on next operation.
```

The other two cases to consider are Windows-to-NFS v4 and NFS v4-to-Windows mappings. A file cannot have both Windows and NFS v4 security at the same time, so this makes these two cases much simpler than the UNIX mappings discussed prior. For one, HNAS does not need to store flags to indicate which mappings are correct and which are not. Mapping between Windows and NFS v4 users and groups only happens when either an NFS v4 client views a file that has a Windows security descriptor, or a Windows client views a file that has an NFS v4 ACL.

Cluster Name Space

The NFS v4 protocol has, built into it, a number of features that were designed specifically to provide a global name space capability, including an *fs_locations4* attribute, which describes the locations of each real file system. The primary goal of this attribute is to report the locations of all replicated copies of a given file system, as well as to redirect a client when a file system is migrated to a different physical node.

HNAS makes use of the *fs_locations4* attributes to *con* clients into thinking a file system has been migrated to cause them to redirect to a different node. For example, with a pseudo file system containing the path */home/span0*, in which *span0* is a link to a real file system, if the client is connected to pnode1 and the file system *span0* is hosted on pnode2, the NFS v4 traffic would then look similar to the following:

```
Client:  -> PUTROOTFH, LOOKUP ("home"), GETFH
pnode1: <- NFS4_OK, NFS4_OK, NFS4_OK (fh = FH_home)
Client:  -> PUTFH (FH_home), LOOKUP ("span0"), GETFH
pnode1: <- NFS4_OK, NFS4_OK, NFS4ERR_MOVED
Client:  -> PUTFH (FH_home), LOOKUP ("span0"), GETATTR (fs_locations4)
```

```
pnode1:  <- NFS4_OK, NFS4_OK, NFS4_OK (location = pnode2:/home/span0)
Client:   -> PUTROOTFH, LOOKUP ("home"), GETFH
pnode2:  <- NFS4_OK, NFS4_OK, NFS4_OK (fh = FH_home)
Client:   -> PUTFH (FH_home), LOOKUP ("span0"), GETFH
pnode2: <- NFS4_OK, NFS4_OK, NFS4_OK
```

This illustrates how the client was conned into thinking the file system *span0* has been migrated from pnode1 to pnode2. It then went to pnode2 and traversed once again through the file system but, on reaching the link, the transition to the real file system was then done automatically as the pseudo and real file systems are hosted on the same pnode.

NFS v4 Management

Management of NFS v4 can be done via the CLI and the web UI.

NFS v4 CLI Commands

The following console commands are used to manage NFS v4:

nfs-export: Used to create and delete exports, which can be accessed over NFS versions 2, 3, or 4.

nfsv4-public-filehandle:
Used to set the NFS v4 public file handle.

nfsv4-compound:
Used to display the **compound** command combinations that the server has seen to date.

nfs-stats: Used to report statistics for NFS versions 2, 3, and 4.

nfs-errors: Used to report the errors that have been returned over NFS versions 2, 3, and 4.

nfs-perf: Used to display performance statistics for NFS versions 2, 3, and 4.

nfsv4-pseudofs-*:
A set of Dev-level commands that can be used to manually update the NFS v4 pseudo file system in case manual intervention is required.

NFS exports are managed via the SMU exactly as they are for versions 2 and 3.

NFS Protocol Selection

The mount options used on the NFS client can select which version of NFS is used. For example, the command to select NFS v3 is *mount -overs=3*. If the **vers** option is omitted, the client asks the server for every version of NFS that it supports and then chooses its preferred version, which will usually be the highest. In addition, a per-EVS setting is available to configure the maximum supported NFS version. This is controlled by way of:

max-supported-nfs-version [2 | 3 | 4]

For example, setting the **max-supported-version** to **4** allows NFS versions 2, 3, or 4 to be used.

NFS v4 Web UI

The web UI has one page for managing protocol specific options, such as setting the public file handle and managing the root and public file handles' access options. The SMU also manages NFS exports (and so the NFS v4 pseudo file system) using the existing NFS export pages.

Secure NFS options (see Kerberos Authentication) cannot be configured from the web UI. However, the NFS export page identifies exports for which secure NFS is enabled.

The NFS stats page has been updated to include NFS v4 statistics.

Hitachi Data Systems

Corporate Headquarters
2845 Lafayette Street
Santa Clara, California 95050-2639
U.S.A.
www.hds.com

Regional Contact Information

Americas
+1 408 970 1000
info@hds.com

Europe, Middle East, and Africa
+44 (0) 1753 618000
info.emea@hds.com

Asia Pacific
+852 3189 7900
hds.marketing.apac@hds.com

MK-92HNAS056-00