

NFSv4 Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 15, 2013

J. Lentini
NetApp
D. Ellard
Raytheon BBN Technologies
R. Tewari
IBM Almaden
C. Lever, Ed.
Oracle Corporation
December 12, 2012

NSDB Protocol for Federated Filesystems
draft-ietf-nfsv4-federated-fs-protocol-15

Abstract

This document describes a filesystem federation protocol that enables file access and namespace traversal across collections of independently administered file servers. The protocol specifies a set of interfaces by which file servers with different administrators can form a file server federation that provides a namespace composed of the filesystems physically hosted on and exported by the constituent file servers.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 15, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1.	Introduction	5
2.	Overview of Features and Concepts	6
2.1.	File-access Protocol	6
2.2.	File-access Client	6
2.3.	Fileserver	6
2.4.	Referral	6
2.5.	Namespace	6
2.6.	Fileset	7
2.7.	Fileset Name (FSN)	7
2.8.	Fileset Location (FSL)	8
2.8.1.	The NFS URI scheme	9
2.8.2.	Mutual Consistency across Fileset Locations	10
2.8.3.	Caching of Fileset Locations	11
2.8.4.	Generating A Referral from Fileset Locations	12
2.9.	Namespace Database (NSDB)	13
2.9.1.	NSDB Client	14
2.10.	Junctions and Referrals	14
2.11.	Unified Namespace and the Root Fileset	15
2.12.	UUID Considerations	15
3.	Examples	16
3.1.	Creating a Fileset and its FSL(s)	16
3.1.1.	Creating a Fileset and an FSN	17
3.1.2.	Adding a Replica of a Fileset	17
3.2.	Junction Resolution	17
3.3.	Example Use Cases for Fileset Annotations	18
4.	NSDB Configuration and Schema	19
4.1.	LDAP Configuration	19
4.2.	LDAP Schema	21
4.2.1.	LDAP Attributes	23
4.2.2.	LDAP Object Classes	37
5.	NSDB Operations	40
5.1.	NSDB Operations for Administrators	41
5.1.1.	Create an FSN	41
5.1.2.	Delete an FSN	42
5.1.3.	Create an FSL	43
5.1.4.	Delete an FSL	46
5.1.5.	Update an FSL	47
5.2.	NSDB Operations for Fileservers	48
5.2.1.	NSDB Container Entry (NCE) Enumeration	48
5.2.2.	Lookup FSLs for an FSN	48
5.3.	NSDB Operations and LDAP Referrals	49
6.	Security Considerations	50
7.	IANA Considerations	51
7.1.	Registry for the fedfsAnnotation Key Namespace	51
7.2.	Registry for FedFS Object Identifiers	51
7.3.	LDAP Descriptor Registration	54

8. Glossary	57
9. References	60
9.1. Normative References	60
9.2. Informative References	61
Appendix A. Acknowledgments	62
Authors' Addresses	63

1. Introduction

A federated filesystem enables file access and namespace traversal in a uniform, secure and consistent manner across multiple independent filesystems within an enterprise or across multiple enterprises.

This document specifies a set of protocols that allow filesystems, possibly from different vendors and with different administrators, to cooperatively form a federation containing one or more federated filesystems. Each federated filesystem's namespace is composed of the filesystems physically hosted on and exported by the federation's filesystems. A federation comprises a common namespace across all its filesystems. A federation can project multiple namespaces and enable clients to traverse each one. A federation can contain an arbitrary number of namespace repositories, each belonging to a different administrative entity, and each rendering a part of the namespace. A federation might also have an arbitrary number of administrative entities responsible for administering disjoint subsets of the filesystems.

Traditionally, building a namespace that spans multiple filesystems has been difficult for two reasons. First, the filesystems that export pieces of the namespace are often not in the same administrative domain. Second, there is no standard mechanism for the filesystems to cooperatively present the namespace. Filesystems may provide proprietary management tools and in some cases an administrator may be able to use the proprietary tools to build a shared namespace out of the exported filesystems. However, relying on vendor-specific proprietary tools does not work in larger enterprises or when collaborating across enterprises because the filesystems are likely to be from multiple vendors or use different software versions, each with their own namespace protocols, with no common mechanism to manage the namespace or exchange namespace information.

The federated filesystem protocols in this document define how to construct a namespace accessible by an NFSv4.0 [3530bis], NFSv4.1 [RFC5661] or newer client and have been designed to accommodate other file access protocols in the future.

The requirements for federated filesystems are described in [RFC5716]. A protocol for administering a filesystem's namespace is described in [FEDFS-ADMIN]. The mechanism for discovering the root of a federated namespace is described in [RFC6641].

In the rest of the document, the term filesystem denotes a filesystem that is part of a federation.

2. Overview of Features and Concepts

2.1. File-access Protocol

A file-access protocol is a network protocol for accessing data. The NFSv4.0 protocol [3530bis] is an example of a file-access protocol.

2.2. File-access Client

File-access clients are standard off-the-shelf network attached storage (NAS) clients that communicate with file servers using a standard file-access protocol.

2.3. Fileserver

File servers are servers that store physical fileset data, or refer file-access clients to other file servers. A file server provides access to its shared filesystem data via a file-access protocol. A file server may be implemented in a number of different ways, including a single system, a cluster of systems, or some other configuration.

2.4. Referral

A referral is a mechanism by which a file server redirects a file-access client to a different file server or export. The exact information contained in a referral varies from one file-access protocol to another. The NFSv4.0 protocol, for example, defines the `fs_locations` attribute for returning referral information to NFSv4.0 clients. The NFSv4.1 protocol introduces the `fs_locations_info` attribute that can return richer referral information to its clients. NFSv4.1 file servers may use either attribute during a referral. Both attributes are defined in [RFC5661].

2.5. Namespace

The goal of a unified namespace is to make all managed data available to any file-access client via the same path in a common filesystem namespace. This should be achieved with minimal or zero configuration on file-access clients. In particular, updates to the common namespace should not require configuration changes to any file-access client.

Filesets, which are the unit of data management, are a set of files and directories. From the perspective of file-access clients, the common namespace is constructed by mounting filesets that are physically located on different file servers. The namespace, which is defined in terms of fileset names and locations, is stored in a set

of namespace repositories, each managed by an administrative entity.

The namespace schema defines the model used for populating, modifying, and querying the namespace repositories. It is not required by the federation that the namespace be common across all file servers. It should be possible to have several independently rooted namespaces.

2.6. Fileset

A fileset is loosely defined as a set of files and the directory tree that contains them. The fileset abstraction is the basic unit of data management. Depending on the configuration, a fileset may be anything from an individual directory of an exported filesystem to an entire exported filesystem on a file server.

2.7. Fileset Name (FSN)

A fileset is uniquely represented by its fileset name (FSN). An FSN is considered unique across a federation. After an FSN is created, it is associated with one or more fileset locations (FSLs) on one or more file servers.

An FSN consists of:

NsdbName: the network location of the Namespace Database (NSDB) node that contains authoritative information for this FSN.

FsnUuid: a UUID (universally unique identifier), conforming to [RFC4122], that is used to uniquely identify an FSN.

FsnTTL: the time-to-live of the FSN's FSL information, in seconds. File servers MUST NOT use cached FSL records after the parent FSN's FsnTTL has expired. An FsnTTL value of zero indicates that file servers MUST NOT cache the results of resolving this FSN.

The NsdbName is not physically stored as an attribute of the record. The NsdbName is obvious to any client that accesses an NSDB, and is indeed authenticated in cases where TLS security is in effect.

The FsnUuid and NsdbName values never change during an FSN's lifetime. However, an FSN's FSL information can change over time, and is typically cached on file servers for performance. More detail on FSL caching is provided in Section 2.8.3.

An FSN record may also contain:

Annotations: name/value pairs that can be interpreted by a fileserver. The semantics of this field are not defined by this document. These tuples are intended to be used by higher-level protocols.

Descriptions: text descriptions. The semantics of this field are not defined by this document.

2.8. Fileset Location (FSL)

An FSL describes one physical location where a complete copy of the fileset's data resides. An FSL contains generic and type specific information which together describe how to access the fileset data at this location. An FSL's attributes can be used by a fileserver to decide which locations it will return to a file-access client.

An FSL consists of:

FslUuid: a UUID, conforming to [RFC4122], that is used to uniquely identify an FSL.

FsnUuid: the UUID of the FSL's FSN.

NsdbName: the network location of the NSDB node that contains authoritative information for this FSL.

The NsdbName is not stored as an attribute of an FSL record for the same reason it is not stored in FSN records.

An FSL record may also contain:

Annotations: name/value pairs that can be interpreted by a fileserver. The semantics of this field are not defined by this document. These tuples are intended to be used by higher-level protocols.

Descriptions: text descriptions. The semantics of this field are not defined by this document.

In addition to the attributes defined above, an FSL record contains attributes that allow a fileserver to construct referrals. For each file-access protocol, a corresponding FSL record subtype is defined.

This document defines an FSL subtype for NFS. An NFS FSL contains information suitable for use in one of the NFSv4 referral attributes (e.g., fs_locations or fs_locations_info, described in [RFC5661]). Section 4.2.2.4 describes the contents of an NFS FSL record.

A fileset also may be accessible by file-access protocols other than NFS. The contents and format of such FSL subtypes are not defined in this document.

2.8.1. The NFS URI scheme

To capture the location of an NFSv4 fileset, we extend the NFS URL scheme specified in [RFC2224]. This extension follows rules for defining Uniform Resource Identifier schemes (see [RFC3986]). In the following text, we refer to this extended NFS URL scheme as an NFS URI.

An NFS URI MUST contain both an authority and a path component. It MUST NOT contain a query component or a fragment component. Use of the familiar "nfs" scheme name is retained.

2.8.1.1. The NFS URI authority component

The rules for encoding the authority component of a generic URI are specified in section 3.2 of [RFC3986]. The authority component of an NFS URI MUST contain the host subcomponent. For globally-scoped NFS URIs, a hostname used in such URIs SHOULD be a fully qualified domain name. See section 3.2.2 of [RFC3986] for rules on encoding non-ASCII characters in hostnames.

An NFS URI MAY contain a port subcomponent as described in section 3.2.3 of [RFC3986]. If this subcomponent is missing, a port value of 2049 is assumed, as specified in [3530bis], Section 3.1.

2.8.1.2. The NFS URI path component

The rules for encoding the path component of a generic URI are specified in section 3.3 of [RFC3986].

According to sections 5 and 6 of [RFC2224], NFS URLs specify a pathname relative to an NFS fileserver's "public filehandle." However, NFSv4 fileservers do not expose a "public filehandle." Instead, NFSv4 pathnames contained in an NFS URI are evaluated relative to the pseudoroot of the fileserver identified in the URI's authority component.

Each component of an NFSv4 pathname is represented as a component4 string (see Section 3.2, "Basic Data Types" of [RFC5661]). The component4 elements of an NFSv4 pathname are encoded as path segments in an NFS URI. NFSv4 pathnames MUST be expressed in an NFS URI as an absolute path. An NFS URI path component MUST NOT be empty. The NFS URI path component starts with a slash ("/") character, followed by one or more path segments which each start with a slash ("/")

character [RFC3986].

Therefore, a double slash always follows the authority component of an NFS URI. For example, the NFSv4 pathname "/" is represented by two slash ("/") characters following an NFS URI's authority component.

The component4 elements of an NFSv4 pathname MUST be prepared using the component4 rules defined in Chapter 12 "Internationalization" of [3530bis] prior to encoding the path component of an NFS URI. As specified in [RFC3986], any non-ASCII characters and any URI-reserved characters, such as the slash ("/") character, contained in a component4 element MUST be represented by URI percent encoding.

2.8.1.3. Encoding an NFS location in an FSL

The path component of an NFS URI encodes the "rootpath" field of the NFSv4 `fs_location4` data type or the "fli_rootpath" of the NFSv4 `fs_locations_item4` data type (see [RFC5661]).

In its "server" field, the NFSv4 `fs_location4` data type contains a list of universal addresses and DNS labels. Each may optionally include a port number. The exact encoding requirements for this information is found in Section 12.6 of [3530bis]. The NFSv4 `fs_locations_item4` data type encodes the same data in its "fli_entries" field (see [RFC5661]). This information is encoded in the authority component of an NFS URI.

The "server" and "fli_entries" fields can encode multiple server hostnames that share the same pathname. An NFS URI, and hence an FSL record, represents only a single hostname and pathname pair. An NFS fileserver MUST NOT combine a set of FSL records into a single `fs_location4` or `fs_locations_item4` unless each FSL record in the set contains the same rootpath value and extended filesystem information.

2.8.2. Mutual Consistency across Fileset Locations

All of the FSLs that have the same FSN (and thereby reference the same fileset) are equivalent from the point of view of access by a file-access client. Different fileset locations for an FSN represent the same data, though potentially at different points in time. Fileset locations are equivalent but not identical. Locations may either be read-only or read-write. Typically, multiple read-write locations are backed by a clustered filesystem while read-only locations are replicas created by a federation-initiated or external replication operation. Read-only locations may represent consistent point-in-time copies of a read-write location. The federation protocols, however, cannot prevent subsequent changes to a read-only

location nor guarantee point-in-time consistency of a read-only location if the read-write location is changing.

Regardless of the type, one file-access client may be referred to a location described by one FSL while another client chooses to use a location described by another FSL. Since updates to each fileset location are not controlled by the federation protocol, it is the responsibility of administrators to guarantee the functional equivalence of the data.

The federation protocols do not guarantee that different fileset locations are mutually consistent in terms of the currency of their data. However, they provide a means to publish currency information so that all file servers in a federation can convey the same information to file-access clients during referrals. Clients use this information to ensure they do not revert to an out-of-date version of a fileset's data when switching between fileset locations. NFSv4.1 provides guidance on how replication can be handled in such a manner. In particular see Section 11.7 of [RFC5661].

2.8.3. Caching of Fileset Locations

To resolve an FSN to a set of FSL records, a fileserver queries the NSDB node named in the FSN for FSL records associated with this FSN. The parent FSN's FsnTTL attribute (see Section 2.7) specifies the period of time during which a fileserver may cache these FSL records.

The combination of FSL caching and FSL migration presents a challenge. For example, suppose there are three file servers named A, B, and C. Suppose further that fileserver A contains a junction J to fileset X stored on fileserver B (see Section 2.10 for a description of junctions).

Now suppose that fileset X is migrated from fileserver B to fileserver C, and the corresponding FSL information for fileset X in the authoritative NSDB is updated.

If fileserver A has cached FSLs for fileset X, a file-access client traversing junction J on fileserver A will be referred to fileserver B, even though fileset X has migrated to fileserver C. If fileserver A had not cached the FSL records, it would have queried the NSDB and obtained the correct location of fileset X.

Typically, the process of fileset migration leaves a redirection on the source fileserver in place of a migrated fileset (without such a redirection, file-access clients would find an empty space where the migrated fileset was, which defeats the purpose of a managed migration).

This redirection might be a new junction that targets the same FSN as other junctions referring to the migrated fileset, or it might be some other kind of directive, depending on the fileserver implementation, that simply refers file-access clients to the new location of the migrated fileset.

Back to our example. Suppose, as part of the migration process, a junction replaces fileset X on fileserver B. Later, either:

- o New file-access clients are referred to fileserver B by stale FSL information cached on fileserver A, or
- o File-access clients continue to access fileserver B because they cache stale location data for fileset X.

In either case, thanks to the redirection, file-access clients are informed by fileserver B that fileset X has moved to fileserver C.

Such redirecting junctions (here, on fileserver B) would not be required to be in place forever. They need to stay in place at least until FSL entries cached on fileserver and locations cached on file-access clients for the target fileset are invalidated.

The FsnTTL field in the FSL's parent FSN (see Section 2.7) specifies an upper bound for the lifetime of cached FSL information, and thus can act as a lower bound for the lifetime of redirecting junctions.

For example, suppose the FsnTTL field contains the value 3600 seconds (one hour). In such a case, administrators SHOULD keep the redirection in place for at least one hour after a fileset migration has taken place, because a referring fileserver might cache the FSL data during that time before refreshing it.

To get file-access clients to access the destination fileserver more quickly, administrators SHOULD set the FsnTTL field of the migrated fileset to a low number or zero before migration begins. It can be reset to a more reasonable number at a later point.

Note that some file-access protocols do not communicate location cache expiry information to file-access clients. In some cases it may be difficult to determine an appropriate lifetime for redirecting junctions because file-access clients may cache location information indefinitely.

2.8.4. Generating A Referral from Fileset Locations

After resolving an FSN to a set of FSL records, the fileserver generates a referral to redirect a file-access client to one or more

of the FSN's FSLs. The fileserver converts the FSL records to a referral format understood by a particular file-access client, such as an NFSv4 `fs_locations` or `fs_locations_info` attribute.

To give file-access clients as many options as possible, the fileserver SHOULD include the maximum possible number of FSL records in a referral. However, the fileserver MAY omit some of the FSL records from the referral. For example, the fileserver might omit an FSL record because of limitations in the file access protocol's referral format.

For a given FSL record, the fileserver MAY convert or reduce the FSL record's contents in a manner appropriate to the referral format. For example, an NFS FSL record contains all the data necessary to construct an `fs_locations_info` attribute, but an `fs_locations_info` attribute contains several pieces of information that are not found in the simpler `fs_locations` attribute. A fileserver constructs entries in an `fs_locations` attribute using the relevant contents of an NFS FSL record.

Whenever the fileserver converts or reduces FSL data, the fileserver SHOULD attempt to maintain the original meaning where possible. For example, an NFS FSL record contains the rank and order information that is included in an `fs_locations_info` attribute (see NFSv4.1's `FSLI4BX_READRANK`, `FSLI4BX_READORDER`, `FSLI4BX_WRITERANK`, and `FSLI4BX_WRITEORDER`). While this rank and order information is not explicitly expressible in an `fs_locations` attribute, the fileserver can arrange the `fs_locations` attribute's locations list based on the rank and order values.

Another example: A single NFS FSL record contains the hostname of one fileserver. A single `fs_locations` attribute can contain a list of fileserver names. An NFS fileserver MAY combine two or more FSL records into a single entry in an `fs_locations` or `fs_locations_info` array only if each FSL record contains the same pathname and extended filesystem information.

Refer to the NFSv4.1 protocol specification [RFC5661], sections 11.9 and 11.10, for further details.

2.9. Namespace Database (NSDB)

The NSDB service is a federation-wide service that provides interfaces to define, update, and query FSN information, FSL information, and FSN to FSL mapping information.

An individual repository of namespace information is called an NSDB node. The difference between the NSDB service and an NSDB node is

analogous to that between the DNS service and a particular DNS server.

Each NSDB node is managed by a single administrative entity. A single administrative entity can manage multiple NSDB nodes.

Each NSDB node stores the definition of the FSNs for which it is authoritative. It also stores the definitions of the FSLs associated with those FSNs. An NSDB node is authoritative for the filesets that it defines.

An NSDB MAY be replicated throughout the federation. If an NSDB is replicated, the NSDB MUST exhibit loose, converging consistency as defined in [RFC3254]. The mechanism by which this is achieved is outside the scope of this document. Many LDAP implementations support replication. These features MAY be used to replicate the NSDB.

2.9.1. NSDB Client

Each NSDB node supports an LDAP [RFC4510] interface. An NSDB client is software that uses the LDAP protocol to access or update namespace information stored on an NSDB node. Details of these transactions are discussed in Section 4.

A domain's administrative entity uses NSDB client software to manage information stored on NSDB nodes.

Fileservers act as an NSDB client when contacting a particular NSDB node to resolve an FSN to a set of FSL records. The resulting location information is then transferred to file-access clients via referrals. Therefore file-access clients never have need to access NSDBs directly.

2.10. Junctions and Referrals

A junction is a point in a particular fileset namespace where a specific target fileset may be attached. If a file-access client traverses the path leading from the root of a federated namespace to the junction referring to a target fileset, it should be able to mount and access the data in that target fileset (assuming appropriate permissions). In other words, a junction can be viewed as a reference from a directory in one fileset to the root of the target fileset.

A junction can be implemented as a special marker on a directory, or by some other mechanism in the fileserver's underlying filesystem. What data is used by the fileserver to represent junctions is not

defined by this document. The essential property is that given a junction, a fileserver must be able to find the FSN for the target fileset.

When a file-access client reaches a junction, the fileserver refers the client to a list of FSLs associated with the FSN targeted by the junction. The client can then mount one of the associated FSLs.

The federation protocols do not limit where and how many times a fileset is mounted in the namespace. Filesets can be nested; a fileset can be mounted under another fileset.

2.11. Unified Namespace and the Root Fileset

The root fileset, when defined, is the top-level fileset of the federation-wide namespace. The root of the unified namespace is the top level directory of this fileset. A set of designated fileservers in the federation can export the root fileset to render the federation-wide unified namespace. When a file-access client mounts the root fileset from any of these designated fileservers it can view a common federation-wide namespace.

2.12. UUID Considerations

To ensure FSN and FSL records are unique across a domain, FedFS employs UUIDs conforming to [RFC4122] to form the distinguished names of LDAP records containing FedFS data (see Section 4.2.2.2).

Because junctions store a tuple containing an FSN UUID and the name and port of an NSDB node, an FSN UUID must be unique only on a single NSDB node. An FSN UUID collision can be detected immediately when an administrator attempts to publish an FSN or FSL by storing it under a specific NSDB Container Entry (NCE) on an authoritative NSDB host.

Note that one NSDB node may store multiple NCEs, each under a different namingContext. If an NSDB node must contain more than one NCE, the federation's admin entity SHOULD provide a robust method for preventing FSN UUID collisions between FSNs that reside on the same NSDB node but under different NCEs.

Because FSLs are children of FSNs, FSL UUIDs must be unique for just a single FSN. As with FSNs, as soon as an FSL is published, its uniqueness is guaranteed.

A fileserver performs the operations described in Section 5.2 as an unauthenticated user. Thus distinguished names of FSN and FSL records, as well as the FSN and FSL records themselves, are required to be readable by anyone who can bind anonymously to an NSDB node.

Therefore FSN and FSL UUIDs should be considered public information.

Version 1 UUIDs contain a host's MAC address and a time stamp in the clear. This gives provenance to each UUID, but attackers can use such details to guess information about the host where the UUID was generated. Security-sensitive installations should be aware that on externally-facing NSDBs, UUIDs can reveal information about the hosts where they are generated.

In addition, version 1 UUIDs depend on the notion that a hardware MAC address is unique across machines. As virtual machines do not depend on unique physical MAC addresses and in any event an administrator can modify the physical MAC address, version 1 UUIDs are no longer considered sufficient.

To minimize the probability of UUIDs colliding, a consistent procedure for generating UUIDs should be used throughout a federation. Within a federation, UUIDs SHOULD be generated using the procedure described for version 4 of the UUID variant specified in [RFC4122].

3. Examples

In this section we provide examples and discussion of the basic operations facilitated by the federated filesystem protocol: creating a fileset, adding a replica of a fileset, resolving a junction, and creating a junction.

3.1. Creating a Fileset and its FSL(s)

A fileset is the abstraction of a set of files and the directory tree that contains them. The fileset abstraction is the fundamental unit of data management in the federation. This abstraction is implemented by an actual directory tree whose root location is specified by a fileset location (FSL).

In this section, we describe the basic requirements for starting with a directory tree and creating a fileset that can be used in the federation protocols. Note that we do not assume that the process of creating a fileset requires any transformation of the files or the directory hierarchy. The only thing that is required by this process is assigning the fileset a fileset name (FSN) and expressing the location of the implementation of the fileset as an FSL.

There are many possible variations to this procedure, depending on how the FSN that binds the FSL is created, and whether other replicas of the fileset exist, are known to the federation, and need to be

bound to the same FSN.

It is easiest to describe this in terms of how to create the initial implementation of the fileset, and then describe how to add replicas.

3.1.1. Creating a Fileset and an FSN

1. Choose the NSDB node that will keep track of the FSL(s) and related information for the fileset.
2. Create an FSN in the NSDB node.

The FSN UUID is chosen by the administrator or generated automatically by administration software. The former case is used if the fileset is being restored, perhaps as part of disaster recovery, and the administrator wishes to specify the FSN UUID in order to permit existing junctions that reference that FSN to work again.

At this point, the FSN exists, but its fileset locations are unspecified.

3. For the FSN created above, create an FSL in the NSDB node that describes the physical location of the fileset data.

3.1.2. Adding a Replica of a Fileset

Adding a replica is straightforward: the NSDB node and the FSN are already known. The only remaining step is to add another FSL.

Note that the federation protocols provide only the mechanisms to register and unregister replicas of a fileset. Fileserver-to-fileserver replication protocols are not defined.

3.2. Junction Resolution

A fileset may contain references to other filesets. These references are represented by junctions. If a file-access client requests access to a fileset object that is a junction, the fileserver resolves the junction to discover one or more FSLs that implement the referenced fileset.

There are many possible variations to this procedure, depending on how the junctions are represented by the fileserver and how the fileserver performs junction resolution.

Step 4 is the only step that interacts directly with the federation protocols. The rest of the steps may use platform-specific

interfaces.

1. The fileserver determines that the object being accessed is a junction.
2. The fileserver does a local lookup to find the FSN of the target fileset.
3. Using the FSN, the fileserver finds the NSDB node responsible for the target FSN.
4. The fileserver contacts that NSDB node and asks for the set of FSLs that implement the target FSN. The NSDB node responds with a (possibly empty) set of FSLs.
5. The fileserver converts one or more of the FSLs to the location type used by the file-access client (e.g., an NFSv4 `fs_locations` attribute as described in [RFC5661]).
6. The fileserver redirects (in whatever manner is appropriate for the client) the client to the location(s).

3.3. Example Use Cases for Fileset Annotations

Fileset annotations can convey additional attributes of a fileset. For example, fileset annotations can be used to define relationships between filesets that can be used by an auxiliary replication protocol. Consider the scenario where a fileset is created and mounted at some point in the namespace. A snapshot of the read-write FSL of that fileset is taken periodically at different frequencies (say, a daily or weekly snapshot). The different snapshots are mounted at different locations in the namespace.

The daily snapshots are considered as different filesets from the weekly ones, but both are related to the source fileset. We can define an annotation labeling the filesets as source and replica. The replication protocol can use this information to copy data from one or more FSLs of the source fileset to all the FSLs of the replica fileset. The replica filesets are read-only while the source fileset is read-write.

This follows the traditional Andrew File System (AFS) model of mounting the read-only volume at a path in the namespace different from that of the read-write volume [AFS].

The federation protocol does not control or manage the relationship among filesets. It merely enables annotating the filesets with user-defined relationships.

Another potential use for annotations is recording references to an FSN. A single annotation containing the number of references could be defined; or multiple annotations, one per reference, could be used to store detailed information on the location of each reference.

As with the replication annotation described above, the maintenance of reference information would not be controlled by the federation protocol. The information would most likely be non-authoritative because the ability to create a junction does not require the authority to update the FSN record. In any event, such annotations could be useful to administrators for determining if an FSN is referenced by a junction.

4. NSDB Configuration and Schema

This section describes how an NSDB is constructed using an LDAP Version 3 [RFC4510] Directory. Section 4.1 describes the basic properties of the LDAP configuration that **MUST** be used in order to ensure compatibility between different implementations. Section 4.2 defines the new LDAP attribute types, the new object types, and specifies how the distinguished name (DN) of each object instance **MUST** be constructed.

4.1. LDAP Configuration

An NSDB is constructed using an LDAP Directory. This LDAP Directory **MAY** have multiple naming contexts. The LDAP Directory's DSA-specific entry (its rootDSE) has a multi-valued `namingContext` attribute. Each value of the `namingContext` attribute is the DN of a naming context's root entry (see [RFC4512]).

For each naming context that contains federation entries (e.g., FSNs and FSLs):

1. There **MUST** be an LDAP entry that is superior to all of the naming context's federation entries in the Directory Information Tree (DIT). This entry is termed the NSDB Container Entry (NCE). The NCE's children are FSNs. An FSN's children are FSLs.
2. The naming context's root entry **MUST** include the `fedfsNsdbContainerInfo` (defined below) as one of its object classes. The `fedfsNsdbContainerInfo`'s `fedfsNcedn` attribute is used to locate the naming context's NCE.

If a naming context does not contain federation entries, it will not contain an NCE and its root entry will not include a `fedfsNsdbContainerInfo` as one of its object classes.

A `fedfsNsdbContainerInfo`'s `fedfsNceDN` attribute contains the Distinguished Name (DN) of the NSDB Container Entry residing under this naming context. The `fedfsNceDN` attribute MUST NOT be empty.

For example, an LDAP directory might have the following entries:

```

-+ [root DSE]
|   namingContext: o=fedfs
|   namingContext: dc=example,dc=com
|   namingContext: ou=system
|
+---- [o=fedfs]
|     fedfsNceDN: o=fedfs
|
+---- [dc=example,dc=com]
|     fedfsNceDN: ou=fedfs,ou=corp-it,dc=example,dc=com
|
+---- [ou=system]
```

In this case, the `o=fedfs` namingContext has an NSDB Container Entry at `o=fedfs`, the `dc=example,dc=com` namingContext has an NSDB Container Entry at `ou=fedfs,ou=corp-it,dc=example,dc=com`, and the `ou=system` namingContext has no NSDB Container Entry.

The NSDB SHOULD be configured with one or more privileged LDAP users. These users are able to modify the contents of the LDAP database. An administrator that performs the operations described in Section 5.1 SHOULD authenticate using the DN of a privileged LDAP user.

It MUST be possible for an unprivileged (unauthenticated) user to perform LDAP queries that access the NSDB data. A fileserver performs the operations described in Section 5.2 as an unprivileged user.

All implementations SHOULD use the same schema. At minimum, each MUST use a schema that includes all objects named in the following sections, with all associated attributes. If it is necessary for an implementation to extend the schema defined here, consider using one of the following ways to extend the schema:

- o Define a `fedfsAnnotation` key and values (see Section 4.2.1.6). Register the new key and values with IANA (see Section 7.1).
- o Define additional attribute types and object classes, then have entries inherit from a class defined in this document and from the

implementation-defined ones.

Given the above configuration guidelines, an NSDB SHOULD be constructed using a dedicated LDAP server. If LDAP directories are needed for other purposes, such as to store user account information, use of a separate LDAP server for those is RECOMMENDED. By using an LDAP server dedicated to storing NSDB records, there is no need to disturb the configuration of any other LDAP directories that store information unrelated to an NSDB.

4.2. LDAP Schema

The schema definitions provided in this document use the LDAP schema syntax defined in [RFC4512]. The definitions are formatted to allow the reader to easily extract them from the document. The reader can use the following shell script to extract the definitions:

```
<CODE BEGINS>
```

```
#!/bin/sh
grep '^ *///' | sed 's?^ */// ??' | sed 's?^ *///$??'
```

```
<CODE ENDS>
```

If the above script is stored in a file called "extract.sh", and this document is in a file called "spec.txt", then the reader can do:

```
<CODE BEGINS>
```

```
sh extract.sh < spec.txt > fedfs.schema
```

```
<CODE ENDS>
```

The effect of the script is to remove leading white space from each line, plus a sentinel sequence of "///".

Code components extracted from this document must include the following license:

```
<CODE BEGINS>
```

```

/// #
/// # Copyright (c) 2010-2012 IETF Trust and the persons identified
/// # as authors of the code. All rights reserved.
/// #
/// # The authors of the code are the authors of
/// # [draft-ietf-nfsv4-federated-fs-protocol-xx.txt]: J. Lentini,
/// # C. Everhart, D. Ellard, R. Tewari, and M. Naik.
/// #
/// # Redistribution and use in source and binary forms, with
/// # or without modification, are permitted provided that the
/// # following conditions are met:
/// #
/// # - Redistributions of source code must retain the above
/// #   copyright notice, this list of conditions and the
/// #   following disclaimer.
/// #
/// # - Redistributions in binary form must reproduce the above
/// #   copyright notice, this list of conditions and the
/// #   following disclaimer in the documentation and/or other
/// #   materials provided with the distribution.
/// #
/// # - Neither the name of Internet Society, IETF or IETF
/// #   Trust, nor the names of specific contributors, may be
/// #   used to endorse or promote products derived from this
/// #   software without specific prior written permission.
/// #
/// # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
/// # AND CONTRIBUTORS "AS IS" 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 COPYRIGHT OWNER OR CONTRIBUTORS 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.
/// #

```

<CODE ENDS>

4.2.1. LDAP Attributes

The following definitions are used below:

- o The "name" attribute described in [RFC4519].
- o The Integer syntax (1.3.6.1.4.1.1466.115.121.1.27) described in [RFC4517].
- o The "integerMatch" rule described in [RFC4517].
- o The Octet String syntax (1.3.6.1.4.1.1466.115.121.1.40) described in [RFC4517].
- o The "octetStringMatch" rule described in [RFC4517].
- o The Boolean syntax (1.3.6.1.4.1.1466.115.121.1.7) described in [RFC4517].
- o The "booleanMatch" rule described in [RFC4517].
- o The "distinguishedNameMatch" rule described in [RFC4517].
- o The DN syntax (1.3.6.1.4.1.1466.115.121.1.12) described in [RFC4517].
- o The "labeledURI" attribute described in [RFC2079].
- o The UUID syntax (1.3.6.1.1.16.1) described in [RFC4530].
- o The UuidMatch rule described in [RFC4530].
- o The UuidOrderingMatch rule described in [RFC4530].

4.2.1.1. fedfsUuid

A fedfsUuid is the base type for all of the universally unique identifiers (UUIDs) used by the federated filesystem protocols.

The fedfsUuid type is based on rules and syntax defined in [RFC4530].

A fedfsUuid is a single-valued LDAP attribute.

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.1 NAME 'fedfsUuid'
///     DESC 'A UUID used by NSDB'
///     EQUALITY uuidMatch
///     ORDERING uuidOrderingMatch
///     SYNTAX 1.3.6.1.1.16.1
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

4.2.1.2. fedfsFsnUuid

A fedfsFsnUuid represents the UUID component of an FSN. An NSDB SHOULD ensure that no two FSNs it stores have the same fedfsFsnUuid.

This attribute is single-valued.

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.4 NAME 'fedfsFsnUuid'
///     DESC 'The FSN UUID component of an FSN'
///     SUP fedfsUuid
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

4.2.1.3. fedfsFsnTTL

A fedfsFsnTTL is the time-to-live in seconds of a cached FSN and its child FSL records. It corresponds to the FsnTTL as defined in Section 2.7. See also Section 2.8.3 for information about caching FSLs. A fedfsFsnTTL MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 4294967295].

This attribute is single-valued.

<CODE BEGINS>


```
///
/// attributetype (
///     1.3.6.1.4.1.31103.1.11 NAME 'fedfsFsnTTL'
///     DESC 'Time to live of an FSN tree'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///
```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.4. fedfsNceDN

A fedfsNceDN stores a distinguished name (DN).

This attribute is single-valued.

<CODE BEGINS>

```
///
/// attributetype (
///     1.3.6.1.4.1.31103.1.14 NAME 'fedfsNceDN'
///     DESC 'NCE Distinguished Name'
///     EQUALITY distinguishedNameMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
///     SINGLE-VALUE
/// )
///
```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.12 is the DN syntax [RFC4517].

4.2.1.5. fedfsFslUuid

A fedfsFslUuid represents the UUID of an FSL. An NSDB SHOULD ensure that no two FSLs it stores have the same fedfsFslUuid.

This attribute is single-valued.

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.8 NAME 'fedfsFslUuid'
///     DESC 'UUID of an FSL'
///     SUP fedfsUuid
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

4.2.1.6. fedfsAnnotation

A fedfsAnnotation contains an object annotation formatted as a key/value pair.

This attribute is multi-valued; an object type that permits annotations may have any number of annotations per instance.

A fedfsAnnotation attribute is a human-readable sequence of UTF-8 characters with no non-terminal NUL characters. The value MUST be formatted according to the following ABNF [RFC5234] rules:

```

ANNOTATION = KEY "=" VALUE
KEY = ITEM
VALUE = ITEM
ITEM = *WSP DQUOTE UTF8-octets DQUOTE *WSP

```

DQUOTE and WSP are defined in [RFC5234], and UTF8-octets is defined in [RFC3629].

The following escape sequences are allowed:

+-----+-----+	
escape sequence	replacement
+-----+-----+	
\\	\
\"	"
+-----+-----+	

A fedfsAnnotation value might be processed as follows:

1. Parse the attribute value according to the ANNOTATION rule, ignoring the escape sequences above.
2. Scan through results of the previous step and replace the escape sequences above.

A `fedfsAnnotation` attribute that does not adhere to this format SHOULD be ignored in its entirety. It MUST NOT prevent further processing of its containing entry.

The following are examples of valid `fedfsAnnotation` attributes:

```
"key1" = "foo"
"another key" = "x=3"
"key-2" = "A string with \" and \\ characters."
"key3"="bar"
```

which correspond to the following key/value pairs:

key	value
key1	foo
another key	x=3
key-2	A string with " and \ characters.
key3	bar

<CODE BEGINS>

```
///
/// attributetype (
///     1.3.6.1.4.1.31103.1.12 NAME 'fedfsAnnotation'
///     DESC 'Annotation of an object'
///     SUP name
/// )
///
```

<CODE ENDS>

4.2.1.7. `fedfsDescr`

A `fedfsDescr` stores an object description. The description MUST be encoded as a UTF-8 string.

This attribute is multi-valued which permits any number of descriptions per entry.

<CODE BEGINS>

```

    ///
    /// attributetype (
    ///     1.3.6.1.4.1.31103.1.13 NAME 'fedfsDescr'
    ///     DESC 'Description of an object'
    ///     SUP name
    /// )
    ///

```

<CODE ENDS>

4.2.1.8. fedfsNfsURI

A fedfsNfsURI stores the host and pathname components of an FSL. A fedfsNfsURI MUST be encoded as an NFS URI (see Section 2.8.1).

The fedfsNfsURI is a subtype of the labeledURI type [RFC2079], with the same encoding rules.

This attribute is single-valued.

<CODE BEGINS>

```

    ///
    /// attributetype (
    ///     1.3.6.1.4.1.31103.1.120 NAME 'fedfsNfsURI'
    ///     DESC 'Location of fileset'
    ///     SUP labeledURI
    ///     SINGLE-VALUE
    /// )
    ///

```

<CODE ENDS>

4.2.1.9. fedfsNfsCurrency

A fedfsNfsCurrency stores the NFSv4.1 fs_locations_server's fls_currency value [RFC5661]. A fedfsNfsCurrency MUST be encoded as an Integer syntax value [RFC4517] in the range [-2147483648, 2147483647].

This attribute is single-valued.

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.103 NAME 'fedfsNfsCurrency'
///     DESC 'up-to-date measure of the data'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.10. fedfsNfsGenFlagWritable

A fedfsNfsGenFlagWritable stores the value of an FSL's NFSv4.1 FSLI4GF_WRITABLE bit [RFC5661]. A value of "TRUE" indicates the bit is set. A value of "FALSE" indicates the bit is not set.

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.104 NAME 'fedfsNfsGenFlagWritable'
///     DESC 'Indicates if the filesystem is writable'
///     EQUALITY booleanMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.7 is the Boolean syntax [RFC4517].

4.2.1.11. fedfsNfsGenFlagGoing

A fedfsNfsGenFlagGoing stores the value of an FSL's NFSv4.1 FSLI4GF_GOING bit [RFC5661]. A value of "TRUE" indicates the bit is set. A value of "FALSE" indicates the bit is not set.

<CODE BEGINS>

```
///
/// attributetype (
///     1.3.6.1.4.1.31103.1.105 NAME 'fedfsNfsGenFlagGoing'
///     DESC 'Indicates if the filesystem is going'
///     EQUALITY booleanMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
///     SINGLE-VALUE
/// )
///
```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.7 is the Boolean syntax [RFC4517].

4.2.1.12. fedfsNfsGenFlagSplit

A fedfsNfsGenFlagSplit stores the value of an FSL's NFSv4.1 FSLI4GF_SPLIT bit [RFC5661]. A value of "TRUE" indicates the bit is set. A value of "FALSE" indicates the bit is not set.

<CODE BEGINS>

```
///
/// attributetype (
///     1.3.6.1.4.1.31103.1.106 NAME 'fedfsNfsGenFlagSplit'
///     DESC 'Indicates if there are multiple filesystems'
///     EQUALITY booleanMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
///     SINGLE-VALUE
/// )
///
```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.7 is the Boolean syntax [RFC4517].

4.2.1.13. fedfsNfsTransFlagRdma

A fedfsNfsTransFlagRdma stores the value of an FSL's NFSv4.1 FSLI4TF_RDMA bit [RFC5661]. A value of "TRUE" indicates the bit is set. A value of "FALSE" indicates the bit is not set.

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.107 NAME 'fedfsNfsTransFlagRdma'
///     DESC 'Indicates if the transport supports RDMA'
///     EQUALITY booleanMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.7 is the Boolean syntax [RFC4517].

4.2.1.14. fedfsNfsClassSimul

A fedfsNfsClassSimul contains the FSL's NFSv4.1 FSLI4BX_CLSIMUL [RFC5661] value. A fedfsNfsClassSimul MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.108 NAME 'fedfsNfsClassSimul'
///     DESC 'The simultaneous-use class of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.15. fedfsNfsClassHandle

A fedfsNfsClassHandle contains the FSL's NFSv4.1 FSLI4BX_CLHANDLE [RFC5661] value. A fedfsNfsClassHandle MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.109 NAME 'fedfsNfsClassHandle'
///     DESC 'The handle class of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.16. fedfsNfsClassFileid

A fedfsNfsClassFileid contains the FSL's NFSv4.1 FSLI4BX_CLFILEID [RFC5661] value. A fedfsNfsClassFileid MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.110 NAME 'fedfsNfsClassFileid'
///     DESC 'The fileid class of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.17. fedfsNfsClassWritever

A fedfsNfsClassWritever contains the FSL's NFSv4.1 FSLI4BX_CLWRITEVER [RFC5661] value. A fedfsNfsClassWritever MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>


```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.111 NAME 'fedfsNfsClassWritever'
///     DESC 'The write-verifier class of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.18. fedfsNfsClassChange

A fedfsNfsClassChange contains the FSL's NFSv4.1 FSLI4BX_CLCHANGE [RFC5661] value. A fedfsNfsClassChange MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.112 NAME 'fedfsNfsClassChange'
///     DESC 'The change class of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.19. fedfsNfsClassReaddir

A fedfsNfsClassReaddir contains the FSL's NFSv4.1 FSLI4BX_CLREADDIR [RFC5661] value. A fedfsNfsClassReaddir MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.113 NAME 'fedfsNfsClassReaddir'
///     DESC 'The readdir class of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.20. fedfsNfsReadRank

A fedfsNfsReadRank contains the FSL's NFSv4.1 FSLI4BX_READRANK [RFC5661] value. A fedfsNfsReadRank MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.114 NAME 'fedfsNfsReadRank'
///     DESC 'The read rank of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.21. fedfsNfsReadOrder

A fedfsNfsReadOrder contains the FSL's NFSv4.1 FSLI4BX_READORDER [RFC5661] value. A fedfsNfsReadOrder MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.115 NAME 'fedfsNfsReadOrder'
///     DESC 'The read order of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.22. fedfsNfsWriteRank

A fedfsNfsWriteRank contains the FSL's FSLI4BX_WRITERANK [RFC5661] value. A fedfsNfsWriteRank MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.116 NAME 'fedfsNfsWriteRank'
///     DESC 'The write rank of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.23. fedfsNfsWriteOrder

A fedfsNfsWriteOrder contains the FSL's FSLI4BX_WRITEORDER [RFC5661] value. A fedfsNfsWriteOrder MUST be encoded as an Integer syntax value [RFC4517] in the range [0, 255].

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.117 NAME 'fedfsNfsWriteOrder'
///     DESC 'The write order of the filesystem'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

4.2.1.24. fedfsNfsVarSub

A fedfsNfsVarSub stores the value of an FSL's NFSv4.1 FSLI4IF_VAR_SUB bit [RFC5661]. A value of "TRUE" indicates the bit is set. A value of "FALSE" indicates the bit is not set.

<CODE BEGINS>

```

///
/// attributetype (
///     1.3.6.1.4.1.31103.1.118 NAME 'fedfsNfsVarSub'
///     DESC 'Indicates if variable substitution is present'
///     EQUALITY booleanMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
///     SINGLE-VALUE
/// )
///

```

<CODE ENDS>

OID 1.3.6.1.4.1.1466.115.121.1.7 is the Boolean syntax [RFC4517].

4.2.1.25. fedfsNfsValidFor

A fedfsNfsValidFor stores an FSL's NFSv4.1 fs_locations_info fli_valid_for value [RFC5661]. A fedfsNfsValidFor MUST be encoded as an Integer syntax value [RFC4517] in the range [-2147483648, 2147483647].

An FSL's parent's fedfsFsntTTL value and its fedfsNfsValidFor value MAY be different.

This attribute is single-valued.

<CODE BEGINS>

```
///
/// attributetype (
///     1.3.6.1.4.1.31103.1.19 NAME 'fedfsNfsValidFor'
///     DESC 'Valid for time'
///     EQUALITY integerMatch
///     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
///     SINGLE-VALUE
/// )
///
```

OID 1.3.6.1.4.1.1466.115.121.1.27 is the Integer syntax [RFC4517].

<CODE ENDS>

4.2.2. LDAP Object Classes

4.2.2.1. fedfsNsdbContainerInfo

A fedfsNsdbContainerInfo describes the location of the NCE.

A fedfsNsdbContainerInfo's fedfsNceDN attribute is REQUIRED.

A fedfsNsdbContainerInfo's fedfsAnnotation and fedfsDescr attributes are OPTIONAL.

<CODE BEGINS>

```
///
/// objectclass (
///     1.3.6.1.4.1.31103.1.1001 NAME 'fedfsNsdbContainerInfo'
///     DESC 'Describes NCE location'
///     SUP top AUXILIARY
///     MUST (
///         fedfsNceDN
///     )
///     MAY (
///         fedfsAnnotation
///         $ fedfsDescr
///     )
///
```

<CODE ENDS>

4.2.2.2. fedfsFsn

A fedfsFsn represents an FSN.

A fedfsFsn's fedfsFsnUuid and fedfsFsnTTL attributes are REQUIRED.

A fedfsFsn's fedfsAnnotation and fedfsDescr attributes are OPTIONAL.

The DN of an FSN is REQUIRED to take the following form: "fedfsFsnUuid=\$FSNUUID,\$NCE", where \$FSNUUID is the UUID of the FSN and \$NCE is the DN of the NCE. Since LDAP requires a DN to be unique, this ensures that each FSN entry has a unique UUID value within the LDAP directory.

<CODE BEGINS>

```

    ///
    /// objectclass (
    ///     1.3.6.1.4.1.31103.1.1002 NAME 'fedfsFsn'
    ///     DESC 'Represents a fileset'
    ///     SUP top STRUCTURAL
    ///     MUST (
    ///         fedfsFsnUuid
    ///         $ fedfsFsnTTL
    ///     )
    ///     MAY (
    ///         fedfsAnnotation
    ///         $ fedfsDescr
    ///     ))
    ///

```

<CODE ENDS>

4.2.2.3. fedfsFsl

The fedfsFsl object class represents an FSL.

The fedfsFsl is an abstract object class. Protocol specific subtypes of this object class are used to store FSL information. The fedfsNfsFsl object class defined below is used to record an NFS FSL's location. Other subtypes MAY be defined for other protocols (e.g., CIFS).

A fedfsFsl's fedfsFslUuid and fedfsFsnUuid attributes are REQUIRED.

A fedfsFsl's fedfsAnnotation, and fedfsDescr attributes are OPTIONAL.

The DN of an FSL is REQUIRED to take the following form:

"fedfsFslUuid=\$FSLUUID,fedfsFsnUuid=\$FSNUUID,\$NCE" where \$FSLUUID is the FSL's UUID, \$FSNUUID is the FSN's UUID, and \$NCE is the DN of the NCE. Since LDAP requires a DN to be unique, this ensures that each FSL entry has a unique UUID value within the LDAP directory.

<CODE BEGINS>

```

    ///
    /// objectclass (
    ///     1.3.6.1.4.1.31103.1.1003 NAME 'fedfsFsl'
    ///     DESC 'A physical location of a fileset'
    ///     SUP top ABSTRACT
    ///     MUST (
    ///         fedfsFslUuid
    ///         $ fedfsFsnUuid
    ///     )
    ///     MAY (
    ///         fedfsAnnotation
    ///         $ fedfsDescr
    ///     ))
    ///

```

<CODE ENDS>

4.2.2.4. fedfsNfsFsl

A fedfsNfsFsl is used to represent an NFS FSL. The fedfsNfsFsl inherits all of the attributes of the fedfsFsl and extends the fedfsFsl with information specific to the NFS protocol.

The DN of an NFS FSL is REQUIRED to take the following form:
 "fedfsFslUuid=\$FSLUUID,fedfsFsnUuid=\$FSNUUID,\$NCE" where \$FSLUUID is the FSL's UUID, \$FSNUUID is the FSN's UUID, and \$NCE is the DN of the NCE. Since LDAP requires a DN to be unique, this ensures that each NFS FSL entry has a unique UUID value within the LDAP directory.

<CODE BEGINS>

```

///
/// objectclass (
///     1.3.6.1.4.1.31103.1.1004 NAME 'fedfsNfsFsl'
///     DESC 'An NFS location of a fileset'
///     SUP fedfsFsl STRUCTURAL
///     MUST (
///         fedfsNfsURI
///         $ fedfsNfsCurrency
///         $ fedfsNfsGenFlagWritable
///         $ fedfsNfsGenFlagGoing
///         $ fedfsNfsGenFlagSplit
///         $ fedfsNfsTransFlagRdma
///         $ fedfsNfsClassSimul
///         $ fedfsNfsClassHandle
///         $ fedfsNfsClassFileid
///         $ fedfsNfsClassWritever
///         $ fedfsNfsClassChange
///         $ fedfsNfsClassReaddir
///         $ fedfsNfsReadRank
///         $ fedfsNfsReadOrder
///         $ fedfsNfsWriteRank
///         $ fedfsNfsWriteOrder
///         $ fedfsNfsVarSub
///         $ fedfsNfsValidFor
///     )
///

```

<CODE ENDS>

5. NSDB Operations

The operations defined by the protocol can be described as several sub-protocols that are used by entities within a federation to perform different roles.

The first of these sub-protocols defines how the state of an NSDB node can be initialized and updated. The primary use of this sub-protocol is by an administrator to add, edit, or delete filesets, their properties, and their fileset locations.

The second of these sub-protocols defines the queries that are sent to an NSDB node in order to perform resolution (or to find other information about the data stored within that NSDB node) and the responses returned by the NSDB node. The primary use of this sub-protocol is by a fileserver in order to perform resolution, but it may also be used by an administrator to query the state of the system.

The first and second sub-protocols are defined as LDAP operations, using the schema defined in the previous section. If each NSDB node is a standard LDAP server, then, in theory, it is unnecessary to describe the LDAP operations in detail, because the operations are ordinary LDAP operations to query and update records. However, we do not require that an NSDB node implement a complete LDAP service, and therefore we define in these sections the minimum level of LDAP functionality required to implement an NSDB node.

The NSDB sub-protocols are defined in Section 5.1 and Section 5.2. The descriptions of LDAP messages in these sections use the LDAP Data Interchange Format (LDIF) [RFC2849]. In order to differentiate constant and variable strings in the LDIF specifications, variables are prefixed by a \$ character and use all upper case characters. For example, a variable named FOO would be specified as \$FOO.

This document uses the term NSDB client to refer to an LDAP client that uses either of the NSDB sub-protocols.

The third sub-protocol defines the queries and other requests that are sent to a fileserver in order to get information from it or to modify the state of the fileserver in a manner related to the federation protocols. The primary purpose of this protocol is for an administrator to create or delete a junction or discover related information about a particular fileserver.

The third sub-protocol is defined as an ONC RPC protocol. The reason for using ONC RPC instead of LDAP is that all fileservers support ONC RPC but some do not support an LDAP Directory server.

The ONC RPC administration protocol is defined in [FEDFS-ADMIN].

5.1. NSDB Operations for Administrators

The admin entity initiates and controls the commands to manage fileset and namespace information. The protocol used for communicating between the admin entity and each NSDB node MUST be the LDAPv3 [RFC4510] protocol.

The names we assign to these operations are entirely for the purpose of exposition in this document, and are not part of the LDAP dialogs.

5.1.1. Create an FSN

This operation creates a new FSN in the NSDB by adding a new fedfsFsn entry in the NSDB's LDAP directory.

A fedfsFsn entry contains a fedfsFsnUuid. The administrator chooses

the fedfsFsnUuid by a process described in Section 2.12). A fedfsFsn entry also contains a fedfsFsnTTL. The fedfsFsnTTL is chosen by the administrator as described in Section 2.8.3.

5.1.1.1. LDAP Request

This operation is implemented using the LDAP ADD request described by the LDIF below.

```
dn: fedfsFsnUuid=$FSNUUID,$NCE
changeType: add
objectClass: fedfsFsn
fedfsFsnUuid: $FSNUUID
fedfsFsnTTL: $TTL
```

For example, if the \$FSNUUID is "e8c4761c-eb3b-4307-86fc-f702da197966", the \$TTL is "300" seconds, and the \$NCE is "o=fedfs" the operation would be:

```
dn: fedfsFsnUuid=e8c4761c-eb3b-4307-86fc-f702da197966,o=fedfs
changeType: add
objectClass: fedfsFsn
fedfsFsnUuid: e8c4761c-eb3b-4307-86fc-f702da197966
fedfsFsnTTL: 300
```

5.1.1.2. Delete an FSN

This operation deletes an FSN by removing a fedfsFsn entry in the NSDB's LDAP directory.

If the FSN entry being deleted has child FSL entries, this function MUST return an error. This ensures that the NSDB will not contain any orphaned FSL entries. A compliant LDAP implementation will meet this requirement since Section 4.8 of [RFC4511] defines the LDAP delete operation to only be capable of removing leaf entries.

Note that the FSN delete function removes the fileset only from a federation namespace (by removing the records for that FSN from the NSDB node that receives this request). The fileset and its data are not deleted. Any junction that has this FSN as its target may continue to point to this non-existent FSN. A dangling reference may be detected when a fileserver tries to resolve a junction that refers to the deleted FSN.

5.1.2.1. LDAP Request

This operation is implemented using the LDAP DELETE request described by the LDIF below.

```
dn: fedfsFsnUuid=$FSNUUID,$NCE
changeType: delete
```

For example, if the \$FSNUUID is "e8c4761c-eb3b-4307-86fc-f702da197966" and \$NCE is "o=fedfs", the operation would be:

```
dn: fedfsFsnUuid=e8c4761c-eb3b-4307-86fc-f702da197966,o=fedfs
changeType: delete
```

5.1.3. Create an FSL

This operation creates a new FSL for the given FSN by adding a new `fedfsFsl` entry in the NSDB's LDAP directory.

A `fedfsFsl` entry contains a `fedfsFslUuid` and `fedfsFsnUuid`. The administrator chooses the `fedfsFslUuid`. The process for choosing the `fedfsFslUuid` is described in Section 2.12. The `fedfsFsnUuid` is the UUID of the FSL's FSN.

The administrator will also set additional attributes depending on the FSL type.

5.1.3.1. LDAP Request

This operation is implemented using the LDAP ADD request described by the LDIF below (NOTE: the LDIF shows the creation of an NFS FSL)

```

dn: fedfsFslUuid=$FSLUUID,fedfsFsnUuid=$FSNUUID,$NCE
changeType: add
objectClass: fedfsNfsFsl
fedfsFslUuid: $FSLUUID
fedfsFsnUuid: $FSNUUID
fedfsNfsURI: nfs://$HOST:$PORT/$PATH
fedfsNfsCurrency: $CURRENCY
fedfsNfsGenFlagWritable: $WRITABLE
fedfsNfsGenFlagGoing: $GOING
fedfsNfsGenFlagSplit: $SPLIT
fedfsNfsTransFlagRdma: $RDMA
fedfsNfsClassSimul: $CLASS_SIMUL
fedfsNfsClassHandle: $CLASS_HANDLE
fedfsNfsClassFileid: $CLASS_FILEID
fedfsNfsClassWritever: $CLASS_WRITEVER
fedfsNfsClassChange: $CLASS_CHANGE
fedfsNfsClassReaddir: $CLASS_READDIR
fedfsNfsReadRank: $READ_RANK
fedfsNfsReadOrder: $READ_ORDER
fedfsNfsWriteRank: $WRITE_RANK
fedfsNfsWriteOrder: $WRITE_ORDER
fedfsNfsVarSub: $VAR_SUB
fedfsNfsValidFor: $TIME
fedfsAnnotation: $ANNOTATION
fedfsDescr: $DESCR

```

For example, if the \$FSNUUID is "e8c4761c-eb3b-4307-86fc-f702da197966", the \$FSLUUID is "ba89a802-41a9-44cf-8447-dda367590eb3", the \$HOST is "server.example.com", \$PORT is "20049", the \$PATH is stored in the file "/tmp/fsl_path", \$CURRENCY is "0" (an up-to-date copy), the FSL is writable, but not going, split, or accessible via RDMA, the simultaneous-use class is "1", the handle class is "0", the fileid class is "1", the write-verifier class is "1", the change class is "1", the readdir class is "9", the read rank is "7", the read order is "8", the write rank is "5", the write order is "6", variable substitution is false, \$TIME is "300" seconds, \$ANNOTATION is ""foo" = "bar"", \$DESC is "This is a description.", and the \$NCE is "o=fedfs", the operation would be (for readability the DN is split into two lines):

```

dn: fedfsFslUuid=ba89a802-41a9-44cf-8447-dda367590eb3,
   fedfsFsnUuid=e8c4761c-eb3b-4307-86fc-f702da197966,o=fedfs
changeType: add
objectClass: fedfsNfsFsl
fedfsFslUuid: ba89a802-41a9-44cf-8447-dda367590eb3
fedfsFsnUuid: e8c4761c-eb3b-4307-86fc-f702da197966
fedfsNfsURI: nfs://server.example.com:20049//tmp/fsl_path
fedfsNfsCurrency: 0
fedfsNfsGenFlagWritable: TRUE
fedfsNfsGenFlagGoing: FALSE
fedfsNfsGenFlagSplit: FALSE
fedfsNfsTransFlagRdma: FALSE
fedfsNfsClassSimul: 1
fedfsNfsClassHandle: 0
fedfsNfsClassFileid: 1
fedfsNfsClassWritever: 1
fedfsNfsClassChange: 1
fedfsNfsClassReaddir: 9
fedfsNfsReadRank: 7
fedfsNfsReadOrder: 8
fedfsNfsWriteRank: 5
fedfsNfsWriteOrder: 6
fedfsNfsVarSub: FALSE
fedfsNfsValidFor: 300
fedfsAnnotation: "foo" = "bar"
fedfsDescr: This is a description.

```

5.1.3.2. Selecting fedfsNfsFsl Values

The fedfsNfsFsl object class is used to describe NFSv4 accessible filesets. For the reasons described in Section 2.8.4, administrators SHOULD choose reasonable values for all LDAP attributes of an NFSv4 accessible fedfsNfsFsl even though some of these LDAP attributes are not explicitly contained in an NFSv4 fs_locations attribute.

When the administrator is unable to choose reasonable values for the LDAP attributes not explicitly contained in an NFSv4 fs_locations attribute, the values in the following table are RECOMMENDED.

LDAP attribute	LDAP value	Notes
fedfsNfsCurrency	negative value	Indicates that the server does not know the currency (see 11.10.1 of [RFC5661]).
fedfsNfsGenFlagWritable	FALSE	Leaving unset is not harmful (see 11.10.1 of [RFC5661]).
fedfsNfsGenFlagGoing	FALSE	NFS client will detect a migration event if the FSL becomes unavailable.
fedfsNfsGenFlagSplit	TRUE	Safe to assume that the FSL is split.
fedfsNfsTransFlagRdma	TRUE	NFS client will detect if RDMA access is available.
fedfsNfsClassSimul	0	0 is treated as non-matching (see 11.10.1 of [RFC5661]).
fedfsNfsClassHandle	0	See fedfsNfsClassSimul note.
fedfsNfsClassFileid	0	See fedfsNfsClassSimul note.
fedfsNfsClassWritever	0	See fedfsNfsClassSimul note.
fedfsNfsClassChange	0	See fedfsNfsClassSimul note.
fedfsNfsClassReaddir	0	See fedfsNfsClassSimul note.
fedfsNfsReadRank	0	Highest value ensures FSL will be tried.
fedfsNfsReadOrder	0	See fedfsNfsReadRank note.
fedfsNfsWriteRank	0	See fedfsNfsReadRank note.
fedfsNfsWriteOrder	0	See fedfsNfsReadRank note.
fedfsNfsVarSub	FALSE	NFSv4 does not define variable substitution in paths.
fedfsNfsValidFor	0	Indicates no appropriate refetch interval (see 11.10.2 of [RFC5661]).

5.1.4. Delete an FSL

This operation deletes an FSL record. The admin requests the NSDB node storing the fedfsFsl to delete it from its database. This operation does not result in fileset data being deleted on any fileserver.

5.1.4.1. LDAP Request

The admin sends an LDAP DELETE request to the NSDB node to remove the FSL.

```
dn: fedfsFslUuid=$FSLUUID,fedfsFsnUuid=$FSNUUID,$NCE
changeType: delete
```

For example, if the \$FSNUUID is "e8c4761c-eb3b-4307-86fc-f702da197966", the \$FSLUUID is "ba89a802-41a9-44cf-8447-dda367590eb3", and the \$NCE is "o=fedfs", the operation would be (for readability the DN is split into two lines):

```
dn: fedfsFslUuid=ba89a802-41a9-44cf-8447-dda367590eb3,
   fedfsFsnUuid=e8c4761c-eb3b-4307-86fc-f702da197966,o=fedfs
changeType: delete
```

5.1.5. Update an FSL

This operation updates the attributes of a given FSL. This command results in a change in the attributes of the fedfsFsl at the NSDB node maintaining this FSL. The values of the fedfsFslUuid and fedfsFsnUuid attributes MUST NOT change during an FSL update.

5.1.5.1. LDAP Request

The admin sends an LDAP MODIFY request to the NSDB node to update the FSL.

```
dn: fedfsFslUuid=$FSLUUID,fedfsFsnUuid=$FSNUUID,$NCE
changeType: modify
replace: $ATTRIBUTE-TYPE
```

For example, if the \$FSNUUID is "e8c4761c-eb3b-4307-86fc-f702da197966", the \$FSLUUID is "ba89a802-41a9-44cf-8447-dda367590eb3", the \$NCE is "o=fedfs", and the administrator wished to change the NFS read rank to 10, the operation would be (for readability the DN is split into two lines):

```
dn: fedfsFslUuid=ba89a802-41a9-44cf-8447-dda367590eb3,
   fedfsFsnUuid=e8c4761c-eb3b-4307-86fc-f702da197966,o=fedfs
changeType: modify
replace: fedfsNfsReadClass
fedfsNfsReadRank: 10
```

5.2. NSDB Operations for Fileservers

5.2.1. NSDB Container Entry (NCE) Enumeration

To find the NCEs for the NSDB nsdb.example.com, a fileserver would do the following:

```
nce_list = empty
connect to the LDAP directory at nsdb.example.com
for each namingContext value $BAR in the root DSE
/* $BAR is a DN */
query for a fedfsNceDN value at $BAR
/*
 * The RFC 4516 LDAP URL for this search would be
 *
 * ldap://nsdb.example.com:389/$BAR?fedfsNceDN??
 *                               (objectClass=fedfsNsdbContainerInfo)
 *
 */
if a fedfsNceDN value is found
    add the value to the nce_list
```

5.2.2. Lookup FSLs for an FSN

Using an LDAP search, the fileserver can obtain all of the FSLs for a given FSN. The FSN's fedfsFsnUuid is used as the search key. The following examples use the LDAP Universal Resource Identifier (URI) format defined in [RFC4516].

To obtain a list of all FSLs for \$FSNUUID on the NSDB named \$NSDBNAME, the following search can be used (for readability the URI is split into two lines):

```
for each $NCE in nce_list
    ldap://$NSDBNAME/fsnUuid=$FSNUUID,$NCE??one?
        (objectClass=fedfsFsl)
```

This search is for the children of the object with DN "fedfsFsnUuid=\$FSNUUID,\$NCE" with a filter for "objectClass=fedfsFsl". The scope value of "one" restricts the search to the entry's children (rather than the entire subtree below the entry) and the filter ensures that only FSL entries are returned.

For example if \$NSDBNAME is "nsdb.example.com", \$FSNUUID is "e8c4761c-eb3b-4307-86fc-f702da197966", and \$NCE is "o=fedfs", the search would be (for readability the URI is split into three lines):


```
ldap://nsdb.example.com/  
    fsnUuid=e8c4761c-eb3b-4307-86fc-f702da197966,o=fedfs  
    ??one?(objectClass=fedfsFsl)
```

The following search can be used to obtain only the NFS FSLs for \$FSNUUID on the NSDB named \$NSDBNAME (for readability the URI is split into two lines):

```
for each $NCE in nce_list  
    ldap://$NSDBNAME/fsnUuid=$FSNUUID,$NCE??one?  
        (objectClass=fedfsNfsFsl)
```

This also searches for the children of the object with DN "fedfsFsnUuid=\$FSNUUID,\$NCE", but the filter for "objectClass = fedfsNfsFsl" restricts the results to only NFS FSLs.

For example if \$NSDBNAME is nsdb.example.com, \$FSNUUID is "e8c4761c-eb3b-4307-86fc-f702da197966", and \$NCE is "o=fedfs", the search would be (for readability the URI is split into three lines):

```
ldap://nsdb.example.com/  
    fsnUuid=e8c4761c-eb3b-4307-86fc-f702da197966,o=fedfs  
    ??one?(objectClass=fedfsNfsFsl)
```

The fileserver will generate a referral based on the set of FSLs returned by these queries using the process described in Section 2.8.4.

5.3. NSDB Operations and LDAP Referrals

The LDAPv3 protocol defines an LDAP referral mechanism that allows an LDAP server to redirect an LDAP client. LDAPv3 defines two types of LDAP referrals: the Referral type defined in Section 4.1.10 of [RFC4511] and the SearchResultReference type defined in Section 4.5.3 of [RFC4511]. In both cases, the LDAP referral lists one or more URIs for services that can be used to complete the operation. In the remainder of this document, the term LDAP referral is used to indicate either of these types.

If an NSDB operation results in an LDAP referral, the NSDB client MAY follow the LDAP referral. An NSDB client's decision to follow an LDAP referral is implementation and configuration dependent. For example, an NSDB client might be configured to follow only those LDAP

referrals that were received over a secure channel, or only those that target an NSDB that supports encrypted communication. If an NSDB client chooses to follow an LDAP referral, the NSDB client MUST process the LDAP referral and prevent looping as described in Section 4.1.10 of [RFC4511].

6. Security Considerations

Both the NFSv4 and LDAPv3 protocols provide security mechanisms. When used in conjunction with the federated filesystem protocols described in this document, the use of these mechanisms is RECOMMENDED. Specifically, the use of RPCSEC_GSS [RFC2203], which is built on the GSS-API [RFC2743], is RECOMMENDED on all NFS connections between a file-access client and fileserver. The "Security Considerations" sections of the NFSv4.0 [3530bis] and NFSv4.1 [RFC5661] specifications contain special considerations for the handling of GETATTR operations for the `fs_locations` and `fs_locations_info` attributes.

NSDB nodes and NSDB clients MUST implement support for TLS [RFC5246], as described in [RFC4513]. For all LDAP connections established by the federated filesystem protocols, the use of TLS is RECOMMENDED.

If an NSDB client chooses to follow an LDAP referral, the NSDB client SHOULD authenticate the LDAP referral's target NSDB using the target NSDB's credentials (not the credentials of the NSDB that generated the LDAP referral). The NSDB client SHOULD NOT follow an LDAP referral that targets an NSDB for which it does not know the NSDB's credentials.

Within a federation, there are two types of components an attacker may compromise: a fileserver and an NSDB.

If an attacker compromises a fileserver, the attacker can interfere with a file-access client's filesystem I/O operations (e.g., by returning fictitious data in the response to a read request) or fabricating a referral. The attacker's abilities are the same regardless of whether or not the federation protocols are in use. While the federation protocols do not give the attacker additional capabilities, they are additional targets for attack. The LDAP protocol described in Section 5.2 SHOULD be secured using the methods described above to defeat attacks on a fileserver via this channel.

If an attacker compromises an NSDB, the attacker will be able to forge FSL information and thus poison the fileserver's referral information. Therefore an NSDB should be as secure as the filesystems which query it. The LDAP operations described in

Section 5 SHOULD be secured using the methods described above to defeat attacks on an NSDB via this channel.

A fileserver binds anonymously when performing NSDB operations. Thus the contents and distinguished names of FSN and FSL records are required to be readable by anyone who can bind anonymously to an NSDB service. Section 2.12 presents the security considerations in the choice of the type of UUID used in these records.

It should be noted that the federation protocols do not directly provide access to filesystem data. The federation protocols only provide a mechanism for building a namespace. All data transfers occur between a file-access client and fileserver just as they would if the federation protocols were not in use. As a result, the federation protocols do not require new user authentication and authorization mechanisms or require a fileserver to act as a proxy for a client.

7. IANA Considerations

7.1. Registry for the fedfsAnnotation Key Namespace

This document defines the fedfsAnnotation key in Section 4.2.1.6. The fedfsAnnotation key namespace is to be managed by IANA. IANA is to create and maintain a new registry entitled "FedFS Annotation Keys". The location of this registry should be under a new heading called "Federated File System (FedFS) Parameters". The URL address can be based off of the new heading name, for example:
<http://www.iana.org/assignments/fedfs-parameters/> ...

Future registrations are to be administered by IANA using the "First Come First Served" policy defined in [RFC5226]. Registration requests MUST include the key (a valid UTF-8 string of any length), a brief description of the key's purpose, and an email contact for the registration. For viewing, the registry should be sorted lexicographically by key. There are no initial assignments for this registry.

7.2. Registry for FedFS Object Identifiers

Using the process described in [RFC2578], one of the authors was assigned the Internet Private Enterprise Numbers range 1.3.6.1.4.1.31103.x. Within this range, the subrange 1.3.6.1.4.1.31103.1.x is permanently dedicated for use by the federated file system protocols. Unassigned OIDs in this range MAY be used for Private Use or Experimental Use as defined in [RFC5226]. New permanent FedFS OID assignments MUST NOT be made using OIDs in

this range.

IANA is to create and maintain a new registry entitled "FedFS Object Identifiers" for the purpose of recording the allocations of FedFS Object Identifiers (OIDs) specified by this document. No future allocations in this registry are allowed.

The location of this registry should be under the heading "Federated File System (FedFS) Parameters", created in Section 7.1. The URL address can be based off of the new heading name, for example:
<http://www.iana.org/assignments/fedfs-parameters/> ...

For viewing, the registry should be sorted numerically by OID value. The contents of the FedFS Object Identifiers registry are given in Table 1.

Note: A descriptor designated below as "historic" reserves an OID used in a past version of the NSDB protocol. Registering such OIDs retains compatibility among existing implementations of the NSDB protocol. This document does not otherwise refer to historic OIDs.

OID	Description	Reference
1.3.6.1.4.1.31103.1.1	fedfsUuid	RFC-TBD1
1.3.6.1.4.1.31103.1.2	fedfsNetAddr	historic
1.3.6.1.4.1.31103.1.3	fedfsNetPort	historic
1.3.6.1.4.1.31103.1.4	fedfsFsnUuid	RFC-TBD1
1.3.6.1.4.1.31103.1.5	fedfsNsdbName	historic
1.3.6.1.4.1.31103.1.6	fedfsNsdbPort	historic
1.3.6.1.4.1.31103.1.7	fedfsNcePrefix	historic
1.3.6.1.4.1.31103.1.8	fedfsFslUuid	RFC-TBD1
1.3.6.1.4.1.31103.1.9	fedfsFslHost	historic
1.3.6.1.4.1.31103.1.10	fedfsFslPort	historic
1.3.6.1.4.1.31103.1.11	fedfsFslTTL	historic
1.3.6.1.4.1.31103.1.12	fedfsAnnotation	RFC-TBD1
1.3.6.1.4.1.31103.1.13	fedfsDescr	RFC-TBD1
1.3.6.1.4.1.31103.1.14	fedfsNceDN	RFC-TBD1
1.3.6.1.4.1.31103.1.15	fedfsFsnTTL	RFC-TBD1
1.3.6.1.4.1.31103.1.100	fedfsNfsPath	historic
1.3.6.1.4.1.31103.1.101	fedfsNfsMajorVer	historic
1.3.6.1.4.1.31103.1.102	fedfsNfsMinorVer	historic
1.3.6.1.4.1.31103.1.103	fedfsNfsCurrency	RFC-TBD1
1.3.6.1.4.1.31103.1.104	fedfsNfsGenFlagWritable	RFC-TBD1
1.3.6.1.4.1.31103.1.105	fedfsNfsGenFlagGoing	RFC-TBD1
1.3.6.1.4.1.31103.1.106	fedfsNfsGenFlagSplit	RFC-TBD1
1.3.6.1.4.1.31103.1.107	fedfsNfsTransFlagRdma	RFC-TBD1
1.3.6.1.4.1.31103.1.108	fedfsNfsClassSimul	RFC-TBD1
1.3.6.1.4.1.31103.1.109	fedfsNfsClassHandle	RFC-TBD1
1.3.6.1.4.1.31103.1.110	fedfsNfsClassFileid	RFC-TBD1
1.3.6.1.4.1.31103.1.111	fedfsNfsClassWritever	RFC-TBD1
1.3.6.1.4.1.31103.1.112	fedfsNfsClassChange	RFC-TBD1
1.3.6.1.4.1.31103.1.113	fedfsNfsClassReaddir	RFC-TBD1
1.3.6.1.4.1.31103.1.114	fedfsNfsReadRank	RFC-TBD1
1.3.6.1.4.1.31103.1.115	fedfsNfsReadOrder	RFC-TBD1
1.3.6.1.4.1.31103.1.116	fedfsNfsWriteRank	RFC-TBD1
1.3.6.1.4.1.31103.1.117	fedfsNfsWriteOrder	RFC-TBD1
1.3.6.1.4.1.31103.1.118	fedfsNfsVarSub	RFC-TBD1
1.3.6.1.4.1.31103.1.119	fedfsNfsValidFor	RFC-TBD1
1.3.6.1.4.1.31103.1.120	fedfsNfsURI	RFC-TBD1
1.3.6.1.4.1.31103.1.1001	fedfsNsdbContainerInfo	RFC-TBD1
1.3.6.1.4.1.31103.1.1002	fedfsFsn	RFC-TBD1
1.3.6.1.4.1.31103.1.1003	fedfsFsl	RFC-TBD1
1.3.6.1.4.1.31103.1.1004	fedfsNfsFsl	RFC-TBD1

Table 1

7.3. LDAP Descriptor Registration

In accordance with Section 3.4 and Section 4 of [RFC4520], the object identifier descriptors defined in this document (listed below) will be registered via the Expert Review process.

Subject: Request for LDAP Descriptor Registration
Person & email address to contact for further information: See
"Author/Change Controller"
Specification: draft-ietf-nfsv4-federated-fs-protocol
Author/Change Controller: IESG (iesg@ietf.org)

Object Identifier: 1.3.6.1.4.1.31103.1.1
Descriptor (short name): fedfsUuid
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.2
Descriptor (short name): fedfsNetAddr
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.3
Descriptor (short name): fedfsNetPort
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.4
Descriptor (short name): fedfsFsnUuid
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.5
Descriptor (short name): fedfsNsdbName
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.6
Descriptor (short name): fedfsNsdbPort
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.7
Descriptor (short name): fedfsNcePrefix
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.8
Descriptor (short name): fedfsFslUuid
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.9
Descriptor (short name): fedfsFslHost
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.10
Descriptor (short name): fedfsFslPort
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.11
Descriptor (short name): fedfsFslTTL
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.12
Descriptor (short name): fedfsAnnotation
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.13
Descriptor (short name): fedfsDescr
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.14
Descriptor (short name): fedfsNceDN
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.15
Descriptor (short name): fedfsFsnTTL
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.100
Descriptor (short name): fedfsNfsPath
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.101
Descriptor (short name): fedfsNfsMajorVer
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.102
Descriptor (short name): fedfsNfsMinorVer
Usage: attribute type (historic)

Object Identifier: 1.3.6.1.4.1.31103.1.103
Descriptor (short name): fedfsNfsCurrency
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.104
Descriptor (short name): fedfsNfsGenFlagWritable
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.105
Descriptor (short name): fedfsNfsGenFlagGoing
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.106
Descriptor (short name): fedfsNfsGenFlagSplit
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.107
Descriptor (short name): fedfsNfsTransFlagRdma
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.108
Descriptor (short name): fedfsNfsClassSimul
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.109
Descriptor (short name): fedfsNfsClassHandle
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.110
Descriptor (short name): fedfsNfsClassFileid
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.111
Descriptor (short name): fedfsNfsClassWritever
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.112
Descriptor (short name): fedfsNfsClassChange
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.113
Descriptor (short name): fedfsNfsClassReaddir
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.114
Descriptor (short name): fedfsNfsReadRank
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.115
Descriptor (short name): fedfsNfsReadOrder
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.116
Descriptor (short name): fedfsNfsWriteRank
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.117
Descriptor (short name): fedfsNfsWriteOrder
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.118
Descriptor (short name): fedfsNfsVarSub
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.119
Descriptor (short name): fedfsNfsValidFor
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.120
Descriptor (short name): fedfsNfsURI
Usage: attribute type

Object Identifier: 1.3.6.1.4.1.31103.1.1001
Descriptor (short name): fedfsNsdbContainerInfo
Usage: object class

Object Identifier: 1.3.6.1.4.1.31103.1.1002
Descriptor (short name): fedfsFsn
Usage: object class

Object Identifier: 1.3.6.1.4.1.31103.1.1003
Descriptor (short name): fedfsFsl
Usage: object class

Object Identifier: 1.3.6.1.4.1.31103.1.1004
Descriptor (short name): fedfsNfsFsl
Usage: object class

8. Glossary

Administrator: A user with the necessary authority to initiate administrative tasks on one or more servers.

Admin Entity: A server or agent that administers a collection of file servers and persistently stores the namespace information.

File-access Client: Standard off-the-shelf network attached storage (NAS) client software that communicates with file servers using a standard file-access protocol.

Federation: A set of file server collections and singleton file servers that use a common set of interfaces and protocols in order to provide to file-access clients a federated namespace accessible through a filesystem access protocol.

Fileserver: A server that stores physical fileset data, or refers file-access clients to other file servers. A file server provides access to its shared filesystem data via a file-access protocol.

Fileset: The abstraction of a set of files and the directory tree that contains them. A fileset is the fundamental unit of data management in the federation.

Note that all files within a fileset are descendants of one directory, and that filesets do not span filesystems.

Filesystem: A self-contained unit of export for a file server, and the mechanism used to implement filesets. The fileset does not need to be rooted at the root of the filesystem, nor at the export point for the filesystem.

A single filesystem MAY implement more than one fileset, if the file-access protocol and the file server permit this.

File-access Protocol: A network filesystem access protocol such as NFSv3 [RFC1813], NFSv4 [3530bis], or CIFS (Common Internet File System) [MS-SMB] [MS-SMB2] [MS-CIFS].

FSL (Fileset Location): The location of the implementation of a fileset at a particular moment in time. An FSL MUST be something that can be translated into a protocol-specific description of a resource that a file-access client can access directly, such as an `fs_locations` attribute (for NFSv4), or a share name (for CIFS).

FSN (Fileset Name): A platform-independent and globally unique name for a fileset. Two FSLs that implement replicas of the same fileset MUST have the same FSN, and if a fileset is migrated from one location to another, the FSN of that fileset MUST remain the same.

Junction: A filesystem object used to link a directory name in the current fileset with an object within another fileset. The server-side "link" from a leaf node in one fileset to the root of another fileset.

Namespace: A filename/directory tree that a sufficiently authorized file-access client can observe.

NSDB (Namespace Database) Service: A service that maps FSNs to FSLs. The NSDB may also be used to store other information, such as annotations for these mappings and their components.

NSDB Node: The name or location of a server that implements part of the NSDB service and is responsible for keeping track of the FSLs (and related info) that implement a given partition of the FSNs.

Referral: A server response to a file-access client access that directs the client to evaluate the current object as a reference to an object at a different location (specified by an FSL) in another fileset, and possibly hosted on another fileserver. The client re-attempts the access to the object at the new location.

Replica: A replica is a redundant implementation of a fileset. Each replica shares the same FSN, but has a different FSL.

Replicas may be used to increase availability or performance. Updates to replicas of the same fileset **MUST** appear to occur in the same order, and therefore each replica is self-consistent at any moment.

We do not assume that updates to each replica occur simultaneously. If a replica is offline or unreachable, the other replicas may be updated.

Server Collection: A set of fileservers administered as a unit. A server collection may be administered with vendor-specific software.

The namespace provided by a server collection could be part of the federated namespace.

Singleton Server: A server collection containing only one server; a stand-alone fileserver.

9. References

9.1. Normative References

- [3530bis] Haynes, T. and D. Noveck, "NFS Version 4 Protocol", draft-ietf-nfsv4-rfc3530bis (Work In Progress), 2010.
- [RFC2079] Smith, M., "Definition of an X.500 Attribute Type and an Object Class to Hold Uniform Resource Identifiers (URIs)", RFC 2079, January 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2203] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, September 1997.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC2849] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.
- [RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.
- [RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.
- [RFC4512] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Directory Information Models", RFC 4512, June 2006.
- [RFC4513] Harrison, R., "Lightweight Directory Access Protocol

(LDAP): Authentication Methods and Security Mechanisms", RFC 4513, June 2006.

- [RFC4516] Smith, M. and T. Howes, "Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator", RFC 4516, June 2006.
- [RFC4517] Legg, S., "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules", RFC 4517, June 2006.
- [RFC4519] Sciberras, A., "Lightweight Directory Access Protocol (LDAP): Schema for User Applications", RFC 4519, June 2006.
- [RFC4520] Zeilenga, K., "Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)", BCP 64, RFC 4520, June 2006.
- [RFC4530] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) entryUUID Operational Attribute", RFC 4530, June 2006.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

9.2. Informative References

- [AFS] Howard, J., "An Overview of the Andrew File System", Proceeding of the USENIX Winter Technical Conference , 1988.
- [FEDFS-ADMIN] Lentini, J., Everhart, C., Ellard, D., Tewari, R., and M. Naik, "Administration Protocol for Federated Filesystems", draft-ietf-nfsv4-federated-fs-admin (Work In Progress), 2010.

- [MS-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol Specification", MS-CIFS 2.0, November 2009.
- [MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", MS-SMB 17.0, November 2009.
- [MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2 Protocol Specification", MS-SMB2 19.0, November 2009.
- [RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, June 1995.
- [RFC2224] Callaghan, B., "NFS URL Scheme", RFC 2224, October 1997.
- [RFC3254] Alvestrand, H., "Definitions for talking about directories", RFC 3254, April 2002.
- [RFC5662] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", RFC 5662, January 2010.
- [RFC5716] Lentini, J., Everhart, C., Ellard, D., Tewari, R., and M. Naik, "Requirements for Federated File Systems", RFC 5716, January 2010.
- [RFC6641] Everhart, C., Adamson, W., and J. Zhang, "Using DNS SRV to Specify a Global File Namespace with NFS Version 4", RFC 6641, June 2012.

Appendix A. Acknowledgments

The authors and editor would like to thank Craig Everhart and Manoj Naik, who were co-authors of an earlier version of this document. In addition, we would like to thank Andy Adamson, Paul Lemahieu, Mario Wurzl, and Robert Thurlow for helping to author this document.

We would like to thank George Amvrosiadis, Trond Myklebust, Howard Chu, and Nicolas Williams for their comments and review.

The editor gratefully acknowledges the IESG reviewers, whose constructive comments helped make this a much stronger document.

Finally, we would like to thank Andy Adamson, Rob Thurlow, and Tom Haynes for helping to get this document out the door.

The `extract.sh` shell script and formatting conventions were first

described by the authors of the NFSv4.1 XDR specification [RFC5662].

Authors' Addresses

James Lentini
NetApp
1601 Trapelo Rd, Suite 16
Waltham, MA 02451
US

Phone: +1 781-768-5359
Email: jlentini@netapp.com

Daniel Ellard
Raytheon BBN Technologies
10 Moulton Street
Cambridge, MA 02138
US

Phone: +1 617-873-8004
Email: dellard@bbn.com

Renu Tewari
IBM Almaden
650 Harry Rd
San Jose, CA 95120
US

Email: tewarir@us.ibm.com

Charles Lever (editor)
Oracle Corporation
1015 Granger Avenue
Ann Arbor, MI 48104
US

Phone: +1 248-614-5091
Email: chuck.lever@oracle.com

