NFSv4 Working Group M. Naik
Internet Draft M. Eshel
Intended Status: Standards Track IBM Almaden
Expires: January 19, 2015 July 18, 2014

Support for File System Extended Attributes in NFSv4
draft-naik-nfsv4-xattrs-01

Abstract

This document proposes extensions to existing NFSv4 operations to allow file extended attributes (here forth also referred to as xattrs) to be manipulated in the protocol. An xattr is a file system feature that allows opaque metadata, not interpreted by the file system, to be associated with files and directories and are supported by many modern file systems. New file attributes are proposed to allow clients to query the server for xattr support, and new operations to get and set xattrs on file system objects.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Table of Contents

1 Introduction 3
1.1 Terminology 3
2 Uses 4
3 Namespaces 5
4 Differences with Named Attributes 5
5 Protocol Enhancements 6
5.1 New Attributes 6
5.1.1 Attribute 82: maxxattrsize 7
5.1.2 Attribute 83: xattrsize 7
5.2 New Operations 7
5.2.1 New definitions 8
5.2.2 Caching 9
5.2.3 GETXATTR - Get extended attributes of a file 9
5.2.4 SETXATTR - Set extended attributes for a file 11
5.2.5 Valid Errors 13
5.3 Extensions to ACE Access Mask Attributes 14
5.4 pNFS Considerations 14
6 Security Considerations 14
7 IANA Considerations 14
8 References 15
8.1 Normative References 15
8.2 Informative References 15
9 Acknowledgements 15
Authors' Addresses 16

1 Introduction

Extended attributes, also called xattrs, are a means to associate opaque metadata with file system objects, typically organized in key/value pairs. They are especially useful when they add information that is not, or cannot be, present in the associated object itself. User-space applications can arbitrarily create, read from, and write to the key/value pairs.

Extended attributes are file system-agnostic; applications use an interface not specific to any file system to manipulate them. Applications do not need to be concerned about how the key/value pairs are stored internally on the underlying file system. All major operating systems provide various flavors of extended attributes. Many user space tools allow xattrs to be included in attributes that need to be preserved when objects are updated, moved or copied.

Extended attributes have long been considered unsuitable for portability because they are inadequately defined and not formally documented by any standard (such as POSIX). However, evidence suggests that xattrs are widely deployed and their support in modern disk-based file systems is fairly universal.

There are no clear indications on how xattrs can be mapped to any existing recommended or optional file attributes defined in RFC 5661 [2]; thereby most NFS client implementations ignore application-specified xattrs. This results in data loss if one copies, over the NFS protocol, a file with xattrs from one file system to another that also supports xattrs.

There is a relatively strong interest in the community in exposing xattrs over NFS despite the shortcomings.

This document discusses why the current NFSv4 named attributes as currently standardized in [2], are unsuitable for representing xattrs, and proposes alternate language, adjustment and protocol mechanisms to support them.

1.1 Terminology

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 RFC 2119 [1].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

2 Uses

Applications can store tracking information in extended attributes. Examples include storing metadata identifying the application that created the file, a tag to indicate when the file was last verified by a data integrity scrubber, or a tag to hold a checksum/crypto hash of the file contents along with the date of that signature. Xattrs can also be used for decorations or annotations. For example, a file downloaded from a web server can be tagged with the URL, which can be convenient if its source has to be determined in the future. Likewise, an email attachment, when saved, can be tagged with the message-id of the email, making it possible to trace the original message.

Applications may need to behave differently when handling files of varying types. For example, file managers, such as GNOME's, offer unique icons, different click behavior, and special lists of operations to perform depending on the file format. This can be achieved by looking at the file extension (Windows), or interpret the type by inspecting it (Unix MIME type). Some file managers generate this information on the fly; others generate the information once and then cache it. Those that cache the information tend to put it in a custom database. The file manager must work to keep this database in sync with the files, which can change without the file manager's knowledge. A better approach is to jettison the custom database and store such metadata in extended attributes: these are easier to maintain, faster to access, and readily accessible by any application [5].

On Mac OSX, applications such as Dropbox, Skydrive (Onedrive), and Google Drive use the extended attribute interface to assign specific tags to folders.

Xattrs can be retrieved and set through system calls or shell commands and generally supported by user-space tools (such as copy tools) that preserve other file attributes.

Extended attributes are supported by many file systems.

In Linux, ext3, ext4, JFS, XFS, Btrfs, among other file systems support extended attributes. The getfattr and setfattr utilities can be used to retrieve and set xattrs. The names of the extended attributes must be prefixed by the name of the category and a dot; hence these categories are generally qualified as name spaces. Currently, four namespaces exist: user, trusted, security and system [5]. Recommendations on how they should be used are published by freedesktop.org [4].

FreeBSD supports extended attributes in two universal namespaces - user and system, although individual file systems are allowed to implement additional namespaces [6].

Solaris 9 and later allows files to have extended attributes, but implements them as "forks", logically represented as files within a hidden directory that is associated with the target file [7].

In the NTFS file system, extended attributes are one of several supported "file streams" [8].

3 Namespaces

Operating systems may define multiple "namespaces" in which xattrs can be set. Namespaces are more than organizational classes; the operating system may enforce different access policies and allow different capabilities depending on the namespace. Linux, for example, defines "security", "system", "trusted" and "user" namespaces, the first three being specific to Linux [4].

Implementations generally agree on the semantics of a "user" namespace, that allows applications to store arbitrary user attribute data with file system objects. Access to this namespace is controlled via the normal file system attributes. As such, getting and setting xattrs from the user namespace can be considered interoperable across platforms and vendor implementations. Attributes from other namespaces are typically platform-specific, but some of them may be generalized into well-defined set of names that promote interoperable implementations. Similarly, attaching the namespace to the attribute key can avoid conflicting use of attributes.

This document does not propose any language to restrict the key names of extended attributes. Future versions, or other related IETF documents, may include additional text to enforce namespace prefix to key names, formalize names of some well-defined xattrs, or impose additional restrictions on the allowed namespaces to user-managed metadata only, in order to prevent the development of non-interoperable implementations. This document, however, does require that the attribute key/value MUST not be interpreted by the NFS clients and servers.

4 Differences with Named Attributes

RFC5661 defines named attributes as opaque byte streams that are associated with a directory or file and referred to by a string name [2]. Named attributes are intended to be used by client applications as a method to associate application-specific data with a regular file or directory. In that sense, xattrs are similar in concept and

use to named attributes, but there are subtle differences.

File systems typically define individual xattrs "get" and "set" operations as being atomic, although collectively they may be independent. Xattrs generally have size limits ranging from a few bytes to several kilobytes; the maximum supported size is not universally defined and is usually restricted by the file system. Similar to ACLs, the amount of xattr data exchanged between the client and server for get/set operations can be considered to fit in a single COMPOUND request, bounded by the channel's negotiated maximum size for requests. Named attributes, on the other hand, are unbounded data streams and do not impose atomicity requirements.

Individual named attributes are analogous to files, and caching of the data for these needs to be handled just as data caching is for ordinary files following close-to-open semantics. Xattrs, on the other hand, impose caching requirements like other file attributes.

Named attributes and xattrs have different semantics and belong to disjoint namespaces. As a result, mapping one to another is, at best, a compromise.

While it should be possible to write guidance about how a client can use the named attribute mechanism to act like xattrs, such as carving out some namespace and specifying locking primitives to enforce atomicity constraints on individual get/set operations, this is problematic. A client application trying to use xattrs through named attributes with a server that supported xattrs directly would get a lower level of service, and could fail to cooperate on a local application running on the server unless the server file system defined its own interoperability constraints. File systems that already implement xattrs and named attributes natively would need additional guidance such as reserving named attribute namespace specifically for implementation purposes.

5 Protocol Enhancements

This section proposes extensions to the NFSv4 protocol operations to allow xattrs to be queried and set by clients. New attributes are added to bitmap4 data type to allow xattr support to be queried. This follows the guidelines specified in [2] with respect to minor versioning. In addition, new operations, namely GETXATTR and SETXATTR, are defined to allow xattr key/value to be queried and set.

5.1 New Attributes

The following RECOMMENDED attributes are proposed for use with

GETATTR. A client can query the server to determine if xattrs are supported, the maximum size of the xattrs that are allowed for a file system object, and the total current size of all the xattrs for a given file system object.

A client may ask for any of these attributes to be returned by setting a bit in the GETATTR request but MUST handle the case where the server does not return them. A client may ask for the set of attributes the server supports and SHOULD NOT request attributes the server does not support.

Name	Id	Data Type	Acc	Defined in
maxxattrsize	82	uint32_t	R	Section 5.1.1
xattrsize	83	uint32_t	R	Section 5.1.2

5.1.1 Attribute 82: maxxattrsize

Maximum size in bytes of all the extended attributes per object that the object's file system supports. If maxxattrsize is 0, the server does not support extended attributes. The protocol does not enforce any limits on the number of keys, the length of a key or the size of a value, that are allowed for a file, as long as the total size is contained by maxxattrsize. The server file system MAY impose additional limits. In addition, the total size of xattrs exchanged between the client and server for get/set operations is limited by the channel's negotiated maximum size for requests and responses.

5.1.2 Attribute 83: xattrsize

The total size of all the extended attributes of this object in bytes. This MUST be less than or equal to maxxattrsize.

5.2 New Operations

Unlike other file system attributes, xattrs can represent disparate metadata most file systems allow disparate metadata to be associated with an object through one or more xattrs, and combining them into a single attribute is unwieldy. As such, adding new attributes to bitmap4 for use in GETATTR and SETATTR is inappropriate to support xattr operations. For example, obtaining the value of a single xattr using the bitmap would require a client implementation to read all the xattrs of the file and find a match for the one requested. Similarly, replacing or deleting a single xattr while keeping the others intact would require a client to read the xattrs first, replacing the existing list with a modified list that excludes the

one to be deleted, and writing out the remaining xattrs. Moreover, distinguishing between creating new and replacing existing xattrs on an object is not possible with the existing bitmap.

Applications need to perform the following operations on a given file's extended attributes [5]:

- Given a file, return a list of all of the file's assigned extended attribute keys.
- Given a file and a key, return the corresponding value.
- Given a file, a key, and a value, assign that value to the key.
- Given a file and a key, remove that extended attribute from the file.

This section introduces two new operations, GETXATTR and SETXATTR, to query and set xattrs. GETXATTR allows listing all the xattrs names, names with values, or querying the value of a single name. SETXATTR allows deleting a single xattr or replacing a few without modifying the rest.

5.2.1 New definitions

The NFS xattr structure is defined as follows:

```
typedef utf8str_cis     xattrname4;
typedef opaque          xattrvalue4<>;

struct xattr4 {
        xattrname4     xa_name;
        xattrvalue4    xa_value;
};
```

Each xattr, defined by xattr4, is a key/value pair. xattrname4 is a UTF-8 string denoting the xattr key name, xattrvalue4 is a variable length string that identifies the values of a specified xattr. The size of the xattr is a combination of the size of its name represented by xattrname4, and its value represented by xattrvalue4. Any regular file or directory may have an array of xattr4, each consisting of a key and associated value. The NFS client or server MUST NOT interpret the contents of xattr4. Similar to ACLs, the client can use the OPEN or ACCESS operations to check access without modifying or reading data or metadata.

Future versions of this document or other related IETF documents may define specific values for xattr key names, or mechanisms for

encoding namespace in xattrname4.

5.2.2 Caching

The caching behavior for extended attributes is similar to other file attributes such as ACLs and is affected by whether OPEN delegation has been granted to a client or not.

When a delegation is in effect, an operation by a second client to a delegated file will cause the server to recall the delegation through a callback. For individual operations, we will describe, under IMPLEMENTATION, when such operations are required to effect a recall. For GETXATTR, see Section 5.2.3.4. For SETXATTR, see Section 5.2.4.4.

When the client does not hold a delegation on the file, xattrs obtained from the server may be cached and clients can use them to avoid subsequent GETXATTR requests. Such caching is write through in that modification to xattrs is always done by means of requests to the server and should not be only done locally. Due to the relative infrequency of xattr updates, it is suggested that all changes be propagated synchronously. The client MUST NOT maintain a cache of modified xattrs.

The result of local caching is that the xattrs maintained on individual clients may not be coherent. Changes made in one order on the server may be seen in a different order on one client and in a third order on another client. In order to manage the incoherency caused by separate operations to obtain xattrs and other file attributes, a client should treat xattrs just like other file attributes with respect to caching as detailed in section 10.6 of RFC 5661 [2]. A client may validate its cached version of xattrs for a file by fetching both the change and time_access attributes and assuming that if the change attribute has the same value as it did when the attributes were cached, then xattrs have not changed.

5.2.3 GETXATTR - Get extended attributes of a file

5.2.3.1 ARGUMENTS

```
enum getxattr_type4 {
        GETXATTR4_LIST   = 0,
        GETXATTR4_ONE    = 1,
        GETXATTR4_ALL    = 2
};
```

```
union getxattr_args4 switch (getxattr_type4 ga_type) {
 case GETXATTR4_ONE:
        xattrname4    ga_name;
 default:
        void;
};

struct GETXATTR4args {
        /* CURRENT_FH: file */
        getxattr_type4     ga_type;
        getxattr_args4     ga_args;
};
```

5.2.3.2 RESULTS

```
union getxattr_res4 switch (getxattr_type4 gr_type) {
 case GETXATTR4_LIST:
        xattrname4    gr_names<>;
 case GETXATTR4_ONE:
        xattrvalue4   gr_value;
 case GETXATTR4_ALL:
        xattr4        gr_xattrs<>;
};

union GETXATTR4res switch (nfsstat4 gr_status) {
 case NFS4_OK:
        getxattr_res4 gr_resok4;
 default:
        void;
};
```

5.2.3.3 DESCRIPTION

The GETXATTR operation will obtain extended attributes for the file system object specified by the current filehandle. The client specifies what kind of xattr information it would like the server to return through the ga_type argument. GETXATTR4_LIST is used to enumerate the set of extended attribute keys assigned to the file. GETXATTR4_ONE returns the value of an extended attribute from the file, given the key. GETXATTR4_ALL returns the key/value pairs for the set of extended attributes assigned to the file.

The server MUST return the xattr key and/or value that the client requests if xattrs are supported by the server for the target file system. If the server does not support xattrs on the target file system, then it MUST NOT return key and/or value and MUST return an error. The server also MUST return an error if it supports xattrs on the target but cannot obtain the requested data. In that case, no

key/value will be returned. If the xattr keys and/or values contained in the server response will exceed the channel's negotiated maximum response size, then the server MUST return NFS4ERR_REP_TOO_BIG in gr_status.

5.2.3.4 IMPLEMENTATION

If there is an OPEN_DELEGATE_WRITE delegation held by another client for the file in question, and size and/or change are among the set of attributes being interrogated in GETATTR, the server can either obtain the actual current value of these attributes from the client holding the delegation by using the CB_GETATTR callback, or revoke the delegation. See Section 18.7.4 of RFC 5661 for details [2]. Consequently, if a client needs to verify the list of extended attributes with the server, it must also query the change attribute of the file with GETATTR. This handling is similar to how a client would revalidate other file attributes such as ACLs.

5.2.4 SETXATTR - Set extended attributes for a file

5.2.4.1 ARGUMENTS

```
enum setxattr_type4 {
        SETXATTR4_CREATE      = 0,
        SETXATTR4_REPLACE     = 1,
        SETXATTR4_DELETE      = 2,
        SETXATTR4_REPLACE_ALL = 3,
        SETXATTR4_DELETE_ALL  = 4
};

union setxattr_args4 switch (setxattr_type4 sa_type) {
 case SETXATTR4_CREATE:
 case SETXATTR4_REPLACE:
 case SETXATTR4_REPLACE_ALL:
        xattr4 sa_xattrs<>;
 case SETXATTR4_DELETE:
        xattrname4 sa_xattrnames<>;
 case SETXATTR4_DELETE_ALL:
        void;
};

struct SETXATTR4args {
        /* CURRENT_FH: file */
        setxattr_args4 sa_args;
};
```

5.2.4.2 RESULTS

```
union setxattr_res4 switch (setxattr_type4 sr_type) {
 case SETXATTR4_CREATE:
 case SETXATTR4_REPLACE:
 case SETXATTR4_DELETE:
        nfsstat4 sr_res<>;
 case SETXATTR4_REPLACE_ALL:
 case SETXATTR4_DELETE_ALL:
        void;
};

union SETXATTR4res switch (nfsstat4 sr_status) {
 case NFS4_OK:
        void;
 default:
        setxattr_res4 sr_array;
};
```

5.2.4.3 DESCRIPTION

The SETXATTR operation changes one or more of the extended attributes of a file system object. The change desired is specified by sr_type. SETXATTR4_CREATE is used to associate the specified values with the extended attribute keys for the file system object specified by the current filehandle. The server MUST return an error if the attribute key already exists. SETXATTR4_REPLACE is also used to set an xattr, but the server MUST return an error if the attribute key does not exist. An application can delete all existing xattrs for a file and replace them with a new set by using SETXATTR4_REPLACE_ALL. SETXATTR4_DELETE can be used to remove the specified xattr keys, if they exist. SETXATTR4_DELETE_ALL removes all the xattr keys for the file.

While the SETXATTR request MAY contain multiple attribute keys and/or values to be changed for a file, this does not impose any atomicity considerations on the server implementation. If the server cannot update all the attributes for the file atomically, it MUST set them in the order specified. In such cases, it is possible that some keys are changed successfully while others encounter errors. To handle this, contained within the SETXATTR results is a "status" field. If any of the change operations incur an error, then the "status" value MUST NOT be NFS4_OK. In this case, the status of the individual change operations is returned in sr_array. If the xattr keys and/or values contained in the client request exceeds the channel's negotiated maximum request size, then the server MUST return NFS4ERR_REQ_TOO_BIG in sr_status.

A successful SETXATTR SHOULD change the file time_modify and change attributes. However, these attributes SHOULD NOT be changed unless the xattrs are changed.

5.2.4.4 IMPLEMENTATION

If the object whose xattrs are being changed has a file delegation that is held by a client other than the one doing the SETXATTR, the delegation(s) must be recalled, and the operation cannot proceed to actually change the xattrs until each such delegation is returned or revoked. In all cases in which delegations are recalled, the server is likely to return one or more NFS4ERR_DELAY errors while the delegation(s) remains outstanding, although it might not do that if the delegations are returned quickly.

5.2.5 Valid Errors

This section contains a table that gives the valid error returns for each new protocol operation. The error code NFS4_OK (indicating no error) is not listed but should be understood to be returnable by all new operations. The error values for all other operations are defined in Section 15.2 of RFC 5661 [2].

Valid Error Returns for Each New Protocol Operation

Operation	Errors
GETXATTR	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_ISDIR, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_NOTDIR, NFS4ERR_PERM, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
SETXATTR	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_ATTRNOTSUPP, NFS4ERR_BADCHAR, NFS4ERR_BADOWNER, NFS4ERR_BAD_RANGE, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_DQUOT, NFS4ERR_EXIST, NFS4ERR_EXPIRED,

	NFS4ERR_FBIG, NFS4ERR_FHEXPIRED,
	NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO,
	NFS4ERR_LOCKED, NFS4ERR_MOVED,
	NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE,
	NFS4ERR_NOSPC, NFS4ERR_NOTDIR,
	NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE,
	NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PERM,
	NFS4ERR_REP_TOO_BIG,
	NFS4ERR_REP_TOO_BIG_TO_CACHE,
	NFS4ERR_REQ_TOO_BIG,
	NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS,
	NFS4ERR_SERVERFAULT, NFS4ERR_STALE,
	NFS4ERR_TOO_MANY_OPS,
	NFS4ERR_UNKNOWN_LAYOUTTYPE,
	NFS4ERR_WRONG_TYPE

5.3 Extensions to ACE Access Mask Attributes

Two new bitmask constants are proposed for the access mask field:

```
const ACE4_GET_XATTRS          = 0x00200000;
const ACE4_SET_XATTRS          = 0x00400000;
```

Permission to get and set the extended attributes of a file. The affected operations are GETXATTR and SETXATTR respectively. No additional granularity of control is implied by these constants for server implementations.

5.4 pNFS Considerations

Both GETXATTR and SETXATTR are sent to the metadata server, which is responsible for coordinating the changes onto the storage devices.

6 Security Considerations

The additions to the NFS protocol for supporting extended attributes do not alter the security considerations of the NFSv4.1 protocol [2].

7 IANA Considerations

There are no IANA considerations in this document. All NFSv4.1 IANA considerations are covered in [2].

8 References

8.1 Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

[3] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", RFC 5662, January 2010.

8.2 Informative References

[4] http://www.freedesktop.org/wiki/CommonExtendedAttributes, "Guidelines for extended attributes".

[5] Love, R., "Linux System Programming: Talking Directly to the Kernel and C Library", O'Reilly Media, Inc., 2007.

[6] http://www.freebsd.org/cgi/man.cgi?query=extattr&sektion=9, "FreeBSD Man Pages - extattr"

[7] http://docs.oracle.com/cd/E19253-01/816-5175/6mbba7f02, "Oracle Man Pages - fsattr"

[8] http://msdn.microsoft.com/en-us/library/windows/desktop/aa364404(v=vs.85).aspx, "File Streams"

9 Acknowledgements

This draft has attempted to capture the discussion on adding xattrs to the NFSv4 protocol from many participants on the IETF NFSv4 mailing list. Valuable input and advice was received from Tom Haynes on the first revision of this draft.

Authors' Addresses

Manoj Naik
IBM Almaden
650 Harry Rd
San Jose, CA 95120

Phone: +1 408-927-1707
Email: mnaik@us.ibm.com

Marc Eshel
IBM Almaden
650 Harry Rd
San Jose, CA 95120

Phone: +1 408-927-1894
Email: eshel@us.ibm.com