NFSv4 D. Noveck, Ed.
Internet-Draft NetApp
Updates: 8881, 7530 (if approved) 9 February 2026
Intended status: Standards Track
Expires: 13 August 2026

ACLs within the NFSv4 Protocols
draft-ietf-nfsv4-acls-update-03

Abstract

This document is part of the set of documents intended to update the description of NFSv4 Minor Version One as part of the rfc8881bis respecification effort for NFSv4.1. It describes the structure and function of NFSv4 Access Control Lists within NFSv4.0 and NFSv4.1. These minor versions and forthcoming ones define ACLs using an ACL structure derived from Windows ACLs.

Support for other ACL approaches such as draft-POSIX ACLs remains an option that could be taken advantage of in later minor versions such as NFSv4.2.

This document describes the structure of these Windows-derived NFSv4 ACLs and their role in the NFSv4 security architecture. While the focus of this document is on the role of these ACLs in providing a more flexible approach to file access authorization than is made available by the POSIX-derived authorization-related attributes, the potential provision of other security-related functionality based on ACLs is covered as well.

Because of the failure of previous specifications to provide a satisfactory description of the authorization semantics of NFSv4 ACLs, this document takes a different approach to many matters while maintaining compatibility with implementations based on previous specifications.

When the resulting document is eventually published as an RFC, it will supersede the descriptions of ACL structure and semantics appearing in existing minor version specification documents for NFSv4.0 and NFSv4.1, thereby updating RFC7530 and RFC8881.

Status of This Memo

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

This Internet-Draft will expire on 13 August 2026.

Copyright Notice

Table of Contents

1. Introduction

This document describes the existing ACL-related features of the NFSv4 protocol, all of which are accessed through the use of a set of OPTIONAL attributes described in Section 5. These attributes provide:

* Additional means of specifying file access authorization constraints that are more flexible than those provided by the authorization model inherited from POSIX, based on the attributes mode, owner, and owner_group.

* A number of security-related facilities separate from authorization that use ACLs to identify sets of actions that might be subject to various forms of monitoring as described in Section 11.

[Consensus Needed (Items #117a, #119a)]: In this document, the relationship among the multiple ACL use models for which potential support was mentioned previously has changed. A core set of functionality, shared in large part with that derived from a subset of the functionality provided by the now-withdrawn draft-POSIX ACLs is presented as the conceptual base of the feature set. Additional sets of features used to provide the functionality needed by clients expecting Windows-like semantics as part of the NFSv4 ACL implementations are considered as OPTIONAL extensions to that core. In addition, it is made clear that support for draft-POSIX ACLs is not yet present in NFSv4.1, but that a subset of that functionality is available.

Because of the unsatisfactory state of current specifications of the ACL features, the work needed to make the description appropriate as part of an updated description of NFSv4.1 is far beyond the level of clarification and cleanup that would normally be expected for a feature that has been part of the NFSv4 protocols for so many years. This includes addressing the following major issues:

* The unacceptable level of laxity regarding the specification of ACL semantics, as described in Section 3.3.

* The unsuccessful attempt to support multiple ACL use models, without explicitly making the choice OPTIONAL, leaving the issue of client-server agreement unaddressed. For details, see Section 6.

* The lack of proper attention to the semantics of draft-POSIX ACLs, which led to the unjustified assumption that such support was essentially already available, when, in fact, it was not. For details, see Section 3.6.

* [Consensus needed (Item #121a, #122a, #123a, #124a)]: The incomplete state of many of the Windows-based ACL features that leaves NFSv4 ACLs insufficient for compatibility with Windows ACLs despite their structural similarities.

Necessary work to address these issues includes the addition of protocol extensions to correct existing defects as described in Section 1.3. For more discussion of the reasons that this situation exists and insights into the work necessary work to provide a satisfactory description, see Section 3.

[Consensus Needed (Item #104a, #105a)]: One important element of a new description of the ACL features is a means by which the client can determine which set of features a server has implemented when the ACL-related attributes are supported. For reasons discussed in more detail in Section 3, that information has not previously been available to NFSv4 clients making it necessary that we provide, for NFSv4.1 and later minor versions, a new OPTIONAL attribute, ACL_Choice, to provide this information, as described in Section 5.9. While we normally do not make additions to the XDR within an existing minor version, we have taken this step now as it is crucial to provide interoperability for this important security-related feature, making it necessary to address this matter as a correctible protocol defect. Addressing such defects is allowed by Section 9 of [RFC8178] in order to enable updates to provide necessary corrections of this sort.

[Consensus Needed (Item #104a), Through end of section]: Given the scale of the problems we now face, it appears that the following additional steps are also necessary to arrive at a satisfactory description:

* A conceptual restructuring of the existing material to consider a small set of UNIX-based features as the conceptual core together with the extensions incorporated in the description as OPTIONAL extensions to that core.

  That restructuring, which is laid out in detail in Section 4.2 will be a necessary part of the basis for the new ACL_Choice attribute.

* [Consensus Needed (Item #117b), Through end of bulleted item]: Clarifying the relationship between the NFSv4 ACL model and the draft-POSIX ACL model described in [Gr端nbacher].

  Because previous specifications did not take proper notice of Many important semantic differences between the draft-POSIX and NFSv4 ACL models, it is now necessary to prepare for the possibility, alluded to in Appendix E, that future minor versions or extensions thereof might provide support for Access Control Lists and Access Control Elements whose structure is different from those presented in this document.

* Addressing the unfortunate consequences of the current semantic laxity, by reducing the range of behavioral diversity to that which is truly necessary. These include situations in which information about Windows semantics requirement are not available, and others in which the new specification cures the previous underspecification going forward and we need to accommodate implementations which were allowed given the lax approach taken the previous approach to ACL specification.

1.1. Relationship to the Overall Security Document

This document is best understood when it is read together with RFCTBD20 which also discusses security features provided that are not connected with ACLs, and which is a complete description in cases in which the OPTIONAL ACL-related attributes are not implemented.

In many cases, the overall security document will have abbreviated descriptions that serve as an introduction to material in this document and reference sections within this document. Similarly, there will be occasions where it is necessary for this document to reference general aspects of NFSv4 security documented in RFCTBD20.

For the most part, these two documents are independent, except for the inter-document references discussed above. However, the following exceptions should be noted:

* Section 1 of RFCTBD20, in its entirety, applies to both documents, even in the absence of explicit inter-document references.

* The terminology defined in Section 4.1 of RFCTBD20 in intended to be used in either document, without an explicit inter-document reference for each use.

* The sections dealing with Security Considerations and IANA Considerations appearing in RFCTBD20, i.e., Sections 18 and 19 of that document apply to the security-related changes being made in the current update as a whole, i.e., to both documents.

* Appendix A of RFCTBD20, in describing the security-related changes made from previous specifications, includes changes made in both this document and the overall security document.

* The Appendices devoted to tracking Consensus Items, i.e., Appendix B of this document and Appendix B of [I-D.dnoveck-nfsv4-security], need to be considered together, even though each appendix applies only to the document in which it appears.

  This is because there are related consensus items in the several documents whose resolutions might affect one another, including some that are the descendants of consensus items affecting material now in multiple documents.

1.2. Relationship to the V4.1 Respecification Effort

This document is a necessary part of the rfc8881bis effort which seeks to provide an updated and corrected specification of NFSv4 Minor Version One. Since ACLs are part of that minor version, a corrected and updated specification of the processing of ACLs would be required as part of that effort.

However, given the wide scope of the ACL model presented in [RFC3530] and subsequent specifications and the wide allowances made in those documents for non-implementation of various elements of that specification, there is necessarily a great deal of uncertainty about the necessary scope of a respecification. Many features might have never been implemented by servers or used by clients and could have been purely speculative when described.

This situation is complicated by the long time between the initial inclusion of this package of features and now. In many cases, there might not be a clear understanding of the gaps between the feature set in the specification and that commonly implemented. In addition, gaps between the specification and implementation of protocol elements which were implemented may not have been understood or were forgotten after implementation work in this area ceased. The practices discussed in Section 3.3 and the attitudes that led to them, make it likely that this would happen without extensive attention being devoted to regularizing this situation.

[Consensus Needed (Items #104b, #105b, #110a, #114a), Through end of section]:

Since we now consider the extensions to the core UNIX ACL model as OPTIONAL features, we have a framework to consider the suitability of these extensions to the protocol. We are now in a different situation from that confronting the working group when the original approach to ACLs was first arrived at. At that time, the idea of conceiving of these extensions as individually selectable OPTIONAL features was not available.

In any case, the realities that make that view appropriate now existed then as well. The way in which the Working Group chose to accommodate the situation have left us with a need to restructure the presentation of ACL features as described in Section 4.2.

It appears that the specification of ACLs to be done as part of the rfc88811bis effort cannot include all of the possible features we are able to identify since:

* Some of these features might not have been implemented due to a lack of interest in the feature.

* There could have been issues exposed by implementation attempts that might have excluded them as candidates for implementation.

* Many of the issues that made these difficult-to-implement in some environments (e.g. UNIX) might have been more refractory than expected.

* Even where none of the above issues exist, it might not be able to research the relevant issues in a reasonable time.

Since it appears impossible to resolve all the questions left unanswered since ACLs were included in the protocol, it appears necessary to focus on the most important subsets in the near term while allowing further development to proceed in later minor versions. The two important feature subsets are:

* The feature set of the core UNIX ACL use model, which is the subset that that servers are essentially required to implement and which clients have been capable of using so far.

  Because of the way previous specifications chose to deal with expected server behavioral diversity (see Section 3.3), there is considerable work involved in identifying probable sources of diversity and making the choices allowed into OPTIONAL features that the client is to be made aware of or uses of SHOULD for which users all allowed to bypass the recommendation as described in Section 4.1.

* [Consensus Needed (Item #117c), Through end of bulleted item]: The subset of extensions to the core UNIX ACL model intended to allow support of NFSv4 ACLs by servers and server-side file systems originally implemented to provide the functionality to support the draft-POSIX ACL model.

  For reasons discussed in more detail in Section 3.6, this subset had been mostly ignored. As a result, clarifying the relationship between draft-POSIX ACLs and NFSv4 ACLs is an essential part of our work, although providing full support for POSIX ACL semantics on either the client or server is unlikely to be doable as part of the respecification effort, for reasons made clear in in Section 6.2. Possible ways of addressing issues outside the respecification effort are discussed in Appendix E.

Work involving means to support the remainder of the extensions within the NFSv4 ACL model will necessarily have a lower priority. Nevertheless, the effort done to identify the extensions will make it possible for clients to use these extensions. Work will be necessary in later minor versions to decide which of these extensions make sense, to remove those that do not, and to make sure clients and server can use these extensions interoperably.

The likely work to be completed as part of the NFSv4.1 respecification is discussed below:

* The work to provide interoperability needs to focus on the core ACL features, each of which is part of the UNIX ACL subset. This will include the interaction of clients depending on this functionality to interoperably interact with servers that support any of the extensions within the more inclusive NFSv4 ACL model.

  The inclusion of two different means of computing modes from ACLs is a flaw that is desirable to remove, but that is probably not possible to eliminate now given the likely existence of servers implementing these two approaches. The possibility of making this a client choice will be deferred to a later minor version as discussed in Appendix F.2

  Unless other issues arise, the current draft is expected to provide this level of interoperability, for those not depending on any of the extensions.

* Consensus Needed (Item #117c)]: For clients depending on the NFSv4 extensions necessary to support the draft-POSIX ACL model, we will need to provide clarification of the semantic differences between NFSv4 ACLs and draft-POSIX ACLs. The result of this clarification is to eliminate unrealistic expectations regarding the ability of mappings between these two models to provide adequate support. For discussion of the basic elements of that support, man of which will not be available in NFSv4.1, see Section 6.2.

* For clients depending on some of the other NFSv4 extensions, the situation is different in that we will try to provide interoperability, but we have to prepared for the possibility that it will not be possible to provide it in time for the completion of the rfc8881bis effort.

  We will give priority to features used by clients for which server implementations exist. While we hope to find no noteworthy behavioral variants, giving us a specification supporting interoperability, that is not guaranteed but we do have the option, as described in Section 12.1, of allowing multiple variants for now and working toward a common approach in later drafts or in subsequent minor versions.

* There will probably be features for which there is no prospect of either client use or server implementations. Although this raises the possibility of eventually deleting such never-implemented proposed features, this will not happen before the end of the respecification effort.

Such feature deletions would only be possible in a new minor version that allows features to be designated mandatory-to-not-implement, as described in Section 12.1.

1.3. Protocol Defects That Might Need to be Corrected

Although normally, protocol respecifications in bis documents do not modify the protocol and focus instead on clarifications, there are provisions within [RFC8178] to provide needed OPTIONAL extensions as a means of correcting protocol defects.

Determining what issues are sufficiently important to justify taking that step is best left to the judgment of the working group. In this draft, I have limited its use to defects that sufficiently compromise the usefulness of the either of two ACL model as to make that model essentially unusable. The following possibilities were considered:

* The lack of any way to determine which of the extensions to the core UNIX ACL model is supported by a particular server. This includes the majority of ACE mask bits, the ACE flags, all of which are effectively OPTIONAL, and various allowed behavioral differences with significant semantic consequences.

  This issue has been addressed by the creation of the OPTIONAL attribute ACL_Choice

* The different handling of partial satisfaction of ALLOW ACEs by draft-POSIX ACLs and non-POSIX ACLs appeared to require a means for clients to make visible their different requirements in this regard.

  The option of defining a new ACE4_NPS_ACE flag to allow the client to request the proper handling of this situation was seriously considered. However, as this issue turned out to be only a small part of the different approaches of the two ACL model in resolving authorization questions, the choice was made to defer resolving this issue in NFSv4.1 and address it later as described in Appendix E.

* The handling of ACL inheritance by means of default ACLs was not supported despite the intention to consider valid servers supporting the draft-POSIX ACL model. From the viewpoint of the Supersession Narrative referred to in Section 3.6, the included ACEs could be dealt with as inherit-only ACEs applying to all object types but that made the necessary adaptation by draft-POSIX ACL clients unduly difficult.

This issue could have been addressed by defining the ACE4_DEFAULT_ACE flag to allow the client to specify that an inherit-only ACE applying to both file and subdirectories would be treated as part of a default ACL, not normally modified when ACLs were set on an object. To complement this, flags were to be added to the aclflag4 word in the sacl and dacl attributes to allow control of which acl(s) were to be modified.

Eventually, it was decided to defer this item as well and address it later by using a separate attribute for default ACLs as described in Appendix E.

* [Consensus Needed (Item #117d), Through end of bulleted item]: Because the existing definition of the OWNER@, GROUP@, and EVERYONE made it impossible to translate reverse-slope modes to ACLs unless DENY ACEs (not part of draft-POSIX ACLs) were supported, it was considered whether it was necessary to define the special users GROUPNOTOWNER@ and OTHERS@ and to provide ACL_Choice support to indicate the presence of handling of these special who values.

  Eventually, it was decided not to do this and instead to warn implementors of the unfortunate consequences of non-support despite its formal status as OPTIONAL. This is despite the previous use of "SHOULD" which was not taken seriously because of its clear contradiction of the intention to allow use of servers supporting draft-POSIX ACLs which have no DENY ACEs. See Section 3.6 for more details.

As a result, the only protocol extension provided via a change to XDR was the addition of the ACL_Choice attribute. See Section 5.9 for further discussion of this attribute, its motivation, and the reason that its absence needs to be treated as a defect, subject to correction as part of the NFSv4.1 respecification effort.

1.4. Effect on Various Minor Versions

Because of the profound underspecification of ACL feature and the associated misuse of RFC2119-defined keywords, it is necessary that this document, when published as an RFC, supersede the existing specification of ACLs for all NFSv4 minor versions ([RFC7530] [RFC8881]).

However, because of the rules relating to minor versions\ changes introduced in [RFC8178] and associated practical considerations regarding minor versions not under active development, the following differences should be noted:

* In the case of minor version zero, despite the supersession noted above, there is no change to the protocol described or to the XDR described by [RFC7531].

  The changes in this document provide a more complete description with more clarity regarding the parts of ACL feature whose implementation is OPTIONAL. As a result, there is no new functionality introduced or compatibility issues expected.

  While there is a conceptual change that allows new forms of ACLs to be defined, this change cannot be realized in minor version zero, since additional attributes to use them cannot be added.

* In the case of minor version one, the description of ACLs in this document supersedes that in [RFC8881]. In addition, it will provide the ACL description for minor version one that applies once the successor of [I-D.ietf-nfsv4-rfc8881bis] is published as an RFC. That document will explicitly refer to the ACL description being published here.

  The changes in this document, while providing a more clear description of the OPTIONAL nature of many elements of the ACL feature, also corrects a number of protocol defects. The correction of protocol defects is explicitly allowed in revised specifications of existing minor versions, as described in Section 9 of [RFC8178]. The limited XDR changes associated with such corrections will be included in the RFC to be derived from [I-D.dnoveck-nfsv4-rfc5662bis]

  Section 1.3 discusses such potential defects, including a number of potential changes to allow for semantics compatible with draft-POSIX ACLs to be incorporated in NFSv4.1. While such changes could be considered correctible defects, it was decided to defer full support for draft-POSIX ACLs to a later minor version, thus leaving the only defect being corrected here to be the lack of a way for the client to determine which OPTIONAL protocol elements were implemented by the server, including elements that were not described that way in previous specifications. This issue was addressed by defining the ACL_Choice attribute.

  Because the ACL_Choice attribute is OPTIONAL, its inclusion cannot result in compatibility issues.

* In the case of later minor versions, the ACL description is inherited from minor version one and the modification described above will apply to those minor version as well.

Because these minor versions are allowed to define new attributes, the conceptual change in this document, allowing the definition on new forms of ACLs could be taken advantage of in minor version two, by adding a protocol extension, as provided for by [RFC8178].

2. Requirements Language

2.1. Keyword Definitions

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 specified in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Special Considerations

Because this document needs to revise previous treatments of its subject, it will need to cite previous treatments of issues that now need to be dealt with in a different way. This will take the form of quotations from documents whose treatment of the subject is being obsoleted, most often direct but sometimes indirect as well.

The items mentioned below can be considered quotations in this context:

* Paragraphs headed "[Previous Treatment] or otherwise annotated as having that status, as described in Section 1 of [I-D.dnoveck-nfsv4-security].

* Sections whose name includes the string "(vestigial)"

Such treatments in quotations will involve use of these BCP14-defined terms in two noteworthy ways:

* The term may have been used inappropriately (i.e not in accord with [RFC2119]), as has been the case for the "RECOMMENDED" attributes, which are, in fact. OPTIONAL.

  In such cases, the surrounding text will make clear that the quoted text does not have a normative effect.

* The term may have been used in accord with [RFC2119], although the resulting normative statement is now felt to be inappropriate.

In such cases, the surrounding text will need to make clear that the text quoted is no longer to be considered normative, often by providing new text that conflicts with the quoted, previously normative, text.

2.3. Use of the Term "SHOULD

[Consensus Needed (Item #4a), Through end of Section]:

The use of the BCP14-defined term "SHOULD" merits particular attention because of:

* The term’s mistaken use common in earlier discussions of matters addressed in this document.

  This problem is dealt with in more detail in Section 3.4.

* How this term will be used in this revised document.

  This subject is dealt with in more detail in Section 4.1

In previous treatments of ACLs, this term was used extensively in contexts in which it was not made clear what might be valid reasons to bypass the implied recommendation, as would be necessary to determine what was and was not allowed.

* In some cases, specific uses of the term were described as "intentional", with the apparent implication that the reason for the use of this term was to allow implementations to ignore the recommended action simply because it was felt to be inconvenient. The effect was that such uses of "SHOULD" were interpreted as "MAY" with the added expectation that implementations bypassing the recommendation should be expected.

  This left uncertain the question of how an "unintentional" use should be interpreted but it made quite clear that this term was not being used in accordance with the intention of BCP14 to have this designation convey something meaningful.

* The majority of uses of this term, presumably unintentional ones, do not seem to be in accord with [RFC2119] in that it is not made clear what might be valid reasons to bypass the recommendation. The only conclusion that can be reached at this point is that the author felt that there might be valid reasons to bypass the recommendation but was unsure if any existed. However, it appears likely that, in most cases the threshold for considering a reason valid in this context were quite low, most likely because it was often assumed that the possible existence of existing software

components (e.g. file systems designed without regard to NFSv4's needs) which made it difficult to conform to the recommendation would constitute a valid reason to bypass the recommendation, the effect on feature interoperability notwithstanding.

As might be expected, this pattern and other cases of excessive deference to server implementation choices created a difficult interoperability situation, which it is now the job of the working group to correct. As part of doing so, we will, as was done in the companion security document RFCTBD20, when using "SHOULD" without reference to specific valid reasons to bypass the recommendation, the understanding is that, in this context, reliance on an earlier specification which allowed behavior now recommended against is a valid reason to continue to behave in that manner even if the allowance was previously communicated through the mistaken use of RFC2119-defined keywords,

Also, with regard to such residual uses of "SHOULD", it needs to be understood that:

* With regard to new server implementations, there are no further valid reasons to bypass the recommendation unless those are explicitly mentioned.

* That when reporting implementation characteristics (e.g. by use of the ACL_Choice attribute) the right to bypass a recommendation that is not to be accepted does not allow an implementer to report the recommendation as being adhered to.

* That clients are under no obligation to accept such variances from these recommendations and MAY, as the implementors judge prudent, decide to not use the ACL feature or to restrict its use to avoid reliance on particular troublesome instance of recommendations being bypassed.

3. Problems to Address

[Consensus Needed (Item #104c, #105c), Through end of section]:

Because of the problems described in Section 3.1, the state of the description of ACLs in current minor version specification documents (i.e., in [RFC7530] and [RFC8881]) makes it inadvisable to use these as a basis for the current specification without major structural revisions.

In order to better understand how this troubling situation came about, see Section 3.2 which outlines the mistakes that led to our current predicament. Later Section 4 and its subsections will

discuss how these revisions were made without imposing unacceptable compatibility issues on implementations that were allowed, intentionally or not, by the older approach.

3.1. Nature of the Existing Problems

[Consensus Needed (Item 104d), Through end of section]:

ACL handling within the NFSv4 protocols is in an unsatisfactory state, characterized by a troubling level of underspecification that:

* Leaves potential implementers unsure about what is needed to implement the feature.

* Provides no reasonable way for the client to determine what part of the existing feature specifications a server implements.

* Puts interoperability essentially beyond reach for anything beyond the core UNIX model.

  This undercuts the usefulness of the extensions needed to provide Windows semantics, which was the reason a more complicated ACL model was incorporated in NFSv4.

3.2. Probable Problem Sources

[Consensus Needed (Item #104d), Through end of section]:

Existing difficulties appear to have arisen from a misguided attempt to accommodate two very different approaches to the issue of providing ACL support within NFSv4, with each one having its own its own strengths and weaknesses:

* An approach based on a subset of ACL work being done for local UNIX file systems. This was consistent with the pattern used in earlier versions of NFS in which the focus the effort was on making file system functions previously accessible locally available over a network.

[Consensus Needed (Items #105d, #117e), Through end of bulleted list]: In this document, we refer to this approach as the core UNIX ACL use model. Although this approach was derived from the now-withdrawn POSIX draft ACLs there are some elements of the latter that are considered distinct and turned out not to be representable within the NFSv4 ACL model. These include the approach to computing Mode based on an ACL, the handling of partial satisfaction of authorization requests using ALLOW ACEs, and the method of supporting ACL inheritance (described as "default ACLs" in the POSIX drafts).

While this approach would have provided a relatively simple path toward implementation on both the client and server sides, it was eventually not chosen.

Unfortunately, it was never made clear what specific gaps caused that result and it still remains unclear to what degree the extensions made available by a more ambitious approach could be used by UNIX clients, even if they could be successfully implemented on the server side.

* A more ambitious approach not so strongly tied to the UNIX ecosystem. This eventually turned out to be one based on Windows ACLs, whose definition contained many elements whose semantics were outside the UNIX approach to file access semantics.

  Although many elements of this more ambitious approach have been implemented, it is not clear that the more ambitious approach, which we refer to as the NFSv4 ACL model was ever implemented by any server.

The more ambitious approach to providing ACL support was added to [RFC3010] as a five-page section that outlined an approach based on Windows ACLs. That section was not really a feature specification did not appear to be attempting to fill that role. Instead, like other ambitious proposals in that RFC, it was a statement of ambition and its approval as a proposed standard did not involve a commitment by implementers to implement that proposal.

When [RFC3530] was published, the ACL section followed the same basic approach. At the same time, there were clear indications that implementations based on the proposal in [RFC3010] might not be possible in many environments. Despite this justified concern, a shift to a more UNIX-oriented approach was not made. Instead, the more ambitious approach continued to be presented as canonical while the specification was revised in an attempt to allow ACLs based on the withdrawn POSIX draft ACLs to be implemented within the XDR established for the original, more ambitious approach.

As things turned out, that attempt failed, since the subset of the NFSv4 ACL model that was treated as required ignored some aspects of the draft-POSIX ACL model.

As a result, the fundamental differences between these two approaches were obscured. Instead of providing explicitly for two distinct approaches, the core UNIX and NFSv4 approaches to ACLs were presented as variant implementations of the same underlying approach, simply because they were expressed using the same XDR. Semantic differences were dealt with only when absolutely necessary and the wider semantic range of the NFSv4 model led to an incorrect approach in which the core UNIX and draft-POSIX ACL model were considered as obstacles to be overcome as part of a supersession narrative.

In order to accommodate these multiple ACL models, the specifications were drafted with a troubling level of semantic laxity, as described in "PROB-lax"/>. As a result, clients had no way to determine what approach a server had implemented. As a result, most clients could only depend on the features of the least ambitious model while these needing features of the more ambitious models could only use them if they approached the issue with no prospect of interoperability and chose particular server implementations based on the features which each supported and adapted each client to each server used. As a result,

* Clients that needed the functionality of draft-POSIX ACLs had to deal with mapping between two sets of ACE masks, without documentation of the process, uncertainty about the partial ALLOW satisfaction required by this model and with no direct support for the default ACL semantic.

  They had to deal with the core UNIX ACL subset which was not a good fit for their needs.

* Clients that needed functionality provided by the NFSv4 extensions had no API for users to use to request those services. In addition, it was impossible for clients to determine whether the particular extension was supported.

3.3. Laxity of Semantic Specifications

[Consensus Needed (Item #104e), Through end of section]:

It appears that the following factors had important roles in motivating an approach that led to the current level of underspecification of ACL semantics:

* A feeling among server implementers that if the specification required facilities not provided by existing filesystem implementations

* A lack of attention to interoperability issues that would arise when filesystem implementation gaps were treated with such solicitude..

* A general unwillingness of working group members to concern themselves with security issues.

  As NFS was developed, the model of NFsv3, where this approach was justifiable, continued to have an effect despite its lack of relevance to the wider semantics of NFSv4.

  Often authentication and authorization were confused, with the unfortunate result that authorization was treated as someone else’s problem since authentication was validly treated as RPC’s problem.

* The effects of the minor-version-oriented development model that continued to have no alternative during the development of NFSv4.

  Given this approach there was an understandable reluctance to raise issues about authorization when these would prevent necessary development of other important protocol areas.

* The common assumption that published documents were essentially perfect, made it difficult to consider that certain matters had not been dealt with properly in documents published years ago.

As we consider the extraordinary semantic laxity regarding ACLs in current specifications, it appears that a number of other factors were involved in arriving at the approach we see reflected in [RFC7530] and [RFC8881].

* A lack of concern about and experience with defining semantics.

  For NFSv3, this was not an issue since all necessary semantics was inherited from POSIX semantics with regard to file access, originally defined to support local operation.

  Unfortunately, the working group’s members with this background were unable to shift their approach to adapt to a very different environment, in which semantic extensions to POSIX, such as named attributes and ACLs required a different approach.

* A lack of concern with authorization issues combined with a tendency to treat these as inherently-server specific.

* A lack of familiarity with the needs of standards-based specifications together the expectation, derived from the history of NFS, that there would be coordinated dominant client and server implementations that would resolve semantic issues at the implementation level.

These resulted in a situation in which important behavioral specifications, important to provide interoperability, were left unresolved by writing the specs in various ways that, appear, in retrospect, to have been ill-advised:

* The use of SHOULD without any attention to a\ provide a clear characterization of the supposed "valid reasons" to bypass the recommendation.

  The common use of this keyword in connection with actions to be taken in implementing ACLs might or might not have affected actual implementations as there was often no reason to bypass the recommendation. Nevertheless, it created a substantial uncertainty about many elements of the specification since it allowed the possibility of variant implementations to develop with clients forced to adapt.

* The designation of certain choices by uses of SHOULD explicitly identified as "intentional".

  Essentially, these uses of SHOULD, had the same effect as if MAY had been used but the fact that MAY was not used obscured the reality that the degree of freedom thereby granted to the server had negative consequences for interoperability.

* In many cases, the specification simply cited distinct possible server behavior, implying that it was the job of the client to adapt to the server behavior.

3.4. Misuse of the RFC2119-Defined Term "SHOULD

The use of the term "SHOULD" has led to extensive difficulties in the specification of ACL semantics because of the lack of clarity of its definition in RFC2119 combined with some problems inherent in the attempt to create an IETF-private vocabulary to be used within Standards-track specifications.

According to RFC2119, "SHOULD" indicates that there "may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before proceeding."

Although the uncertainty inherent in use of the word "may" in the phrase "may exist valid reasons" raises a red flag, this is not the essence of the problem. if the above had said "there are valid reasons" instead, use of this term would still not have a clear meaning unless it were made clear, as it rarely is, what the putatively valid reasons were and there was clarity regarding the "full implications".

Furthermore, if one were to endeavor to understand and carefully weigh the implications, this would happen when implementing the specification, often long after the specification was published. As a result, although the required presence of the statement in Section 2.1 is easily verified and enforced, its accuracy and clarity is almost never a concern during review and the author, having little choice in the drafting, has no reason to look beyond trying to choose among the small set of choices deemed "normative" and make clear recommendations, as an author might normally expect to provide.

Within previous specifications related to ACLs, this confused situation resulted in numerous uses of the term "SHOULD" where it was unclear what the author considered might be valid reasons to bypass the recommendation, including some designated as "intentional" where is it is hard to avoid the conclusion that the author felt that a server implementor’s feeling that the recommended choice was uncongenial was to be treated as valid.

Because of this approach’s devastating affect on interoperability, a different approach will be followed in this revised specification, as described in Section 4.1.

3.5. Problems Deriving from ACE Mask Granularity Differences

In previous specifications, it was assumed that, given the finer granularity of the ACE mask word compared to the three POSIX permission bits the difference between these two approaches could be usefully summarized as the result of different levels of granularity, with each mask bit considered either equivalent to a POSIX permission bit or a finer-grained subdivision of one such bit.

While this approach is fairly close to reality so that, when not looked at in detail, it seems to correspond to reality, there are a number of issues where it does not apply or omits important details that were never addressed in previous specifications:

* When one set of mask bits is a finer-grained replacement of a single POSIX permission bit, it is not always the case, it as is often assumed, that the choice of mask tit straightforward.

  While this is often the case (e.g., ACE4_ADD_FILE, ACE4_ADD_SUBDIRECTORY, a ACE4_DELETE_CHILD), there are other cases for which there are complexities that need to be explained in detail.

  The cases ACE4_WRITE_DATA and ACE4_APPEND_DATA, while clearly finer-grained correlates of the POSIX W-bit, need, but never had a clear description of how this distinction could be made in practice.

* There no finer-granularity variants of the R-bit (corresponding to ACE4_READ_DATA) or the X-bit (corresponding to ACE4_EXECUTE).

* A number of mask bits clearly do not control a subset of the permitted actions associated with a permission bit and instead cover a subset of the actions permitted to the owner of the file.

* A significant number of mask bits cover other sort of permission bits, including those never denied in the POSIX authorization model and other actions outside the existing POSIX semantics.

Because of the previous assumption that the differences between the POSIX permission bits and the ACE mask could be addressed in this way, it was clear that a major conceptual shift was necessary. For information about that shift and the way the above issues were dealt with, see Section 4.

3.6. Problems in the Handling of Draft POSIX ACLs

[Consensus Needed (Item #117f), Entire section]:

In previous specifications, it was assumed that, because the draft-POSIX ACL model provided a coarser-grained approach to authorization, support for this model could be provided by servers supporting the NFSv4 ACL model. Essentially, the draft-POSIX model was considered to be as a submodel. In this document, we refer to this unjustified assumption as the "Supersession Narrative" since it was driven by the idea the NFSv4 finer-grained NFSv4 ACL model could supersede the draft-POSIX model.

In addition, it was assumed that servers built to support draft-POSIX ACLs could serve in this role while providing support for a subset of NFSv4 ACLs. It was assumed that, for a clearly defined subset of NFSv4 ACLs, a simple translation to the NFSv4 format could provide

for effective implementation of something sufficiently close to the draft-POSIX ACL model to be useful. The assumption that it was possible to use such servers to support NFSv4 ACLs ignored many of the fundamental differences between the two ACL models. Analysis of the differences between the semantics of the NFSv4 ACL model and the draft-POSIX ACL model established that this was unachievable for the reasons discussed below.

* The handling of inheritance within the NFSv4 ACL model, was not usable in the draft-POSIX ACL context, because the draft-POSIX ACL concept of default ACLs had no correlate within the NFSv4 ACL model.

* The one case in which a specific semantic difference was mentioned, with the presumed draft-POSIX ACL semantics explicitly made essentially OPTIONAL (although confusingly described by an "intentional SHOULD") did not accomplish the intended purpose because it did not accurately reflect draft-POSIX ACL semantics.

* The role of the acl mask within the draft-POSIX ACL approach was essentially ignored.

  This made it extremely difficult to extend the NFSv4 ACL model to incorporate draft-POSIX ACLs because important information needed to support it was excluded from the already-defined ACE/ACL structures.

* Most important was the two different approaches to authorization checking. For draft-POSIX ACLs a single ACE is selected based on the requesting principal, while in the NFSv4 ACL model any number of applicable ACEs can allow or deny the requested operation. This restricts the set of translatable ACLs but is even more troublesome in that it makes the translation dependent on group membership. This requires the translator to have extensive knowledge in order to be able to determine when two named groups or a name user and named groups might intersect and means that a translation can be rendered invalid post facto in response to group membership changes.

Given the above, the assumption that servers built to support draft-POSIX ACLs could support a submodel of the NFSv4 ACL model appears to be grossly mistaken. It appears that the neglect of the details of the defined semantics noted previously caused the semantic requirements of draft-POSIX ACLs to be essentially ignored.

In addition, The corresponding assumption regarding client support neglected much of the semantics of draft-POSIX ACLs. As a result, only a subset of the possible ACLs within the draft-POSIX ACL model were expressible within NFSv4 ACL model.

Given the pervasive lack of attention to semantic details, it seems to have been assumed that the draft-POSIX ACL model could be handled in the framework of the Supersession Narrative discussed above, leaving others with the job of providing true support for draft-POSIX ACLs.

4. Revised approach to ACL Specification

This necessary conceptual revision consists of:

* A conceptual restructuring of the NFSv4 ACL feature described in Section 6.

  Instead of considering the full Windows-based ACL model as canonical, with gaps in the support for various ACE bits, treated as due to differences in permission granularity, the core UNIX ACL model is presented as the always present feature core, with provisions for feature optionality or acceptable behavioral variants dealt with as described in Section 4.2.

* A new approach to the use of the term "SHOULD" that is consistent wit BCP-14 but avoids the difficulties discussed in Section 3.4.

  This new approach and its implications for this document and further development of ACL-related features is discussed in Section 4.1.

4.1. Revised Use of the RFC2119-Defined Term "SHOULD"

In light of the author’s qualms about the use of the term "SHOULD" laid out in Section 3.4, the use of this term will be significantly different than in previous specifications, while staying within the bounds of the statement in Section 2.1:

* This term will only be used if it is the case that a valid reason to bypass the recommendation actually exists.

* The specification will make clear what that reason is.

* The specification will provide meaningful guidance with regard to helping the reader understand the "full implications" of a decision to bypass the recommendation.

* That the phrase "before proceeding" allows those who relied in the past on a previous a specification to not conform to the current recommendation because proceeding as the implementer did before was acceptable then and its potential subsequent invalidity is not now considered significant.

  The Working Group retains the option in a future minor version for which behavioral changes can be specified, to restate things so previously valid behavior become invalid.

One of the common expected uses of "SHOULD" in the respecification would be to accommodate cases in which multiple behavioral variants were allowed, avoiding undue deference to future server implementor's desire for their own convenience, whether this was done using "SHOULD" or otherwise.

In its decisions regarding the possible continued validity of multiple behavioral variants currently allowed, the working group will need to balance:

* The needs of clients and severs to continue to interact as they have in the past.

* The desire to provide a common definition that is widely adhered to establish a basis for interoperability among clients and servers developing and extending ACL-related features.

The behaviors allowed temporarily by this sort of use of the term "SHOULD" are in a position analogous to implementation features typically described as having been "deprecated". Although deprecation per se has no role in the IETF model since a given behavior is either allowed or not (tertium non datur), the possibility of change in a future minor version my serve as motivation to move toward behaviors that are RECOMMENDED.

4.2. Treating Extensions/Variants as OPTIONAL

For reasons much the same as those that motivated the analogous requirements in [RFC8178], the client is faced with the possibility of dealing with implementations that might behave differently in ways that could require the client to adapt or to decide that the feature implementation provided is not usable.

The basic way such information is conveyed is through the ACL_Choice attribute, described in Section 5.9.

5. ACL-based Authorization-related Attributes

[Author Aside: (Items #14a, #15a...), Applies to entire top-level section]: The treatment of the various ACL-based attributes in the included subsections replaces the corresponding sections in earlier documents, in which the attribute descriptions were not consolidated in one place and were disbursed among a number of top-level sections. Where it has been necessary to make significant changes, the annotations for those changes, including author asides and proposed text, appear here while vestigial text that is now superseded has not been brought forward.

The per-object attributes Acl, Dacl, and Sacl all contain an ACL object as described in Sections 7 and 8 and their subsections.

5.1. Definitions to Support ACL-related attributes

The definition of the acemask4 appears below.

This needs to be provided earlier than other definitions because they are used directly in the definition of ACL-related attributes as well as within the definition of the ACE structure in Section 8.

```
<CODE BEGINS>
typedef uint32_t        acemask4;
<CODE ENDS>
```

The definition of the individual bits within mask words of type acemask4 appears in Section 8.3

5.2. Designation of the Absence of an ACL

[Consensus Needed (Item #118a), Through end of section]:

ACL-related attributes are represented as or contain an array of ACEs. While it is possible to treat zero-length arrays of ACEs within this framework and consider them as ACLs, these are treated differently for the following reasons:

* There needs to be some way to indicate the absence of an ACL, rather than a particular ACL value, when interrogating or setting such attributes as acl, dacl, and sacl.

  Intuitively, a zero-length ACE array could serve this purpose.

* There is no good reason to interpret such an empty array as an ACL, since the natural result, an ACL denying all access, would not serve useful purpose.

Given the relationship between mode and acl-related attributes, such universal denial would most easily be expressed using a mode whose nine low-order bits were all zero.

Such an interpretation would make it hard to delete an existing dacl, while retaining the access specified by the mode attribute.

5.3. New Errors Applicable to Setting of ACL attributes

[Consensus Needed (Item #120a), Through end of section]: The errors listed below have been added to NFSv4.1 to deal with situations not previously dealt with. These all relate to ACLs that exceed the limits established for the sizes of ACLs to be stored or fetched.

The error have code in an isolated range unlikely to be used by extensions to NFSv4.2 and subsequent minor versions.

```
<CODE BEGINS>
NFS4ERR_ACLST_POORFIT  = 400000000,
NFS4ERR_ACLST_NOFIT    = 400000001,
NFS4ERR_ACLST_TOOBIG   = 400000002,
NFS4ERR_ACLFT_TOOBIG   = 400000003
<CODE ENDS>
```

These error are used as discussed below:

* NFS4ERR_ACLST_POORFIT is returned when an ACL being set cannot be stored and its ACE count is above the limit guarantee but not above the routine limit.

  The client is entitled to conclude that the ACL was not of excessive size but that it could not be stored because of weaknesses in the filesystem’s ACL implementation

* NFS4ERR_ACLST_NOFIT is returned when an ACL being set cannot be stored and its ACE count is above the routine limit, but is of a size that could, under other circumstances, be stored.

  This informs the client that it took a chance in speculatively storing such a large ACE, but that the attempt failed.

* NFS4ERR_ACLST_TOOBIG is returned when the ACL being set contains so many ACEs and that there is no chance that ACLs with the same number of ACEs could be stored by the filesystem.

  The client is entitled to conclude that that filesystem is incapable of storing ACLs of that size.

* NFS4ERR_ACLFT_TOOBIG is returned when an ACL is being interrogated is so large that the filesystem cannot return it. In this case, it is likely that the ACL was not set using NFSv4 and was stored using local file access or another remote file access protocol.

  This error can also being returned in cases in which an ACL is not being fetched such as the specification of an ACL-related attribute when using VERIFY and NVERIFY. In such situations, the ACL specified could not have been fetched and this error is returned.

5.4. Table of ACL-related Attributes

The following table summarizes all the ACL-related attributes, including:

* Attributes to support ACL-based authentication: Acl, Dacl.

* Attributes to provide other security-related services: Sacl.

* Attributes to provide information regarding the level of ACL support provided: ACL_Choice, ACL_Support.

Name	Id	Ver	Data Type	Acc	Defined in:
Acl	12	4.0	nfsace4<>	R W	Section 5.7
ACL_Choice	87	4.1(bis)	nfsaclc4	R	Section 5.9
ACL_Support	13	4.0	uint32_t	R	Section 5.8
Dacl	58	4.1	nfsacl41	R W	Section 5.10
Sacl	59	4.1	nfsacl41	R W	Section 5.11

Table 1

5.5. Size Limitations for ACL-related attributes

[Consensus Needed (Item #120b), Through end of section]: Although the XDR definitions of the contents of the acl, dacl, and sacl attributes include unlimited variable-length arrays of ACEs, there will inevitably be practical limits on the size of these arrays. The limits discussed here arise at two different levels of implementation:

* Limits on the size of the ACLs that can be stored, typically imposed by the server-side filesystem.

* Limits that are imposed by the protocol use to transfer ACL to and from the client. Of particular importance are size constraints imposed by limits that arise from the characteristics of the session used to effect the ACL transfer.

ACL size limits imposed by the filesystem are made visible to the client as part of the ACL_Choice attribute. See Section 12.4.

Although a single limit, specified in terms of an ACE count often suffices, there are some possible filesystem structure that justify the more complicated arrangement described below.

* There is a limit guarantee expressed as an ACE count. Any ACL containing this count of ACEs is guaranteed storable by the filesystem, unless prevented by situations such as the unavailability of space or IO errors.

* To provide greater flexibility for file systems that do not provide for a large fixed limit because the ACL space is shared with other unrelated functions or because certain ACEs take a large amount of space, an additional routine limit is also provided.

  The routine limit is also expressed as an ACE count. ACEs that contain fewer ACEs than the routine limit but more than the limit guarantee, while not guaranteed to be storable but are expected to be routinely storable with rejections limited to a very small fraction of attempts, expected to be less than or equal to one failure in a thousand attempts.

* To accommodate situations in which larger ACEs can be stored using local file access or through use of other remote file access protocols, a fetch limit can also be provided to allow ACLs not storable usingNFSv4 to be fetched as NFSv4 attribute values.

  The fetch limit is expressed as a byte count for the value as fetched including the actual size of variable-length data items and XDR overhead.

It is important to understand that there are filesystems that are, for various reasons, unable to provide a reasonable size limit guarantee or even a limit guarantee above zero. Although, the NFSv4 protocol does not restrict the use of ACL s in such circumstances, it is important to understand how the usefulness of ACL support for filesystems unable to provide a limit guarantee of at least sixteen is likely to affected.

* When the ACL_Choice attribute is supported, and the information described in Section 12.4 is provided, the client has knowledge of the inability of the associated server to store certain ACLs and MAY treat a needed ACL feature as unusable and treat such situations as it would have if the ACL-related attributes were not supported.

  Furthermore, when this information is not provided, the client MAY also treat the lack of notice of reliable support for ACLs of the size it needs as a valid reason to not use ACLs on such filesystems.

  While it might appear to some that the statement above undercuts the optionality of the ACL_Choice feature, it should be noted that it is common to impose requirements for multiple OPTIONAL features to be implemented together and that the above approach is the least intrusive way of allowing clients to use ACL features on servers that provide useful support for it, allowing client implementors and users the necessary freedom to exercise their judgment regarding these matters.

* It is often the case that operation with very small limit guarantees has an experimental character and that the use of the adaptations described in Appendices D.2.2 and F.4 might be required for eventual routine use.

* In many cases, filesystem implementation changes might be needed to provide ACL support suitable for NFSv4 use.

  While size limits might be usable in cases in which users can become aware of and adjust to size limit that result from tight restrictions on data structure size or their sharing in order to provide efficient access in all supported cases, remote access and the need for reliable interoperability may make the needs for the remote access case substantially different.

  Given those needs, implementers need to consider the possibility of placing ACLs, in a less-than-optimal location, allowing large ACLs to be dealt with reduced performance, rather than rejecting them.

The handling of size limits connected with the protocol data transfer infrastructure (e.g., those deriving from session-based size limits are substantially different

* The errors discussed in Section 5.3 are not returned in these cases.

  This because requests using large ACLs (i.e., SETATTR, VERIFY, NVERIFY) cannot be formed and issued while attempts to fetch them using GSATTR will be responded to without error by simply not setting the associated attribute mask bit in the response.

* Recovery from these troublesome situations is possible and is discussed in [I-D.ietf-nfsv4-rfc8881bis].

  Certain unusual COMPOUNDs ca encounter issues due to reply cache size limits. For discussion of how to avoid such situation, advice regarding avoiding encountering problems with such limits is discussed in. [I-D.ietf-nfsv4-rfc8881bis].

5.6. Types of ACLs

The ACL allows authorization schemes outside those conforming to the POSIX approach to be specified and applied to file objects. This provides additional flexibility in a number of ways:

1. There may be multiple users or sets of users assigned different privileges to address cases in which the appropriate privilege assignments do not conform to the POSIX model in that they are different for users in the same group or different for two groups outside the owning group.

   ACLs support this additional flexibility by allowing an array of Access Control Entries, each of which specifies handling for a user or user group.

2. For particular users or sets of users, the set of operations to be allowed might not be expressible using the three bits provided by POSIX as supplemented by special privileges for operations reserved to file owner.

NFSv4 ACLs, as described in Section 7, address both issues by defining, within the Access Control Entry, a large set of distinct privilege bits, modeled on those provided by Windows ACLs.

ACLs based on a the core subset of NFSv4 ACLs (i.e., the parts which are not OPTIONAL whenever ACL-related attributes are supported make a more limited change to the POSIX authorization model and are

represented by the same sorts of structures as NFSv4 ACLs, although there are restrictions imposed by the UNIX ACL model. These ACLs are based on a subset of the draft-POSIX ACL model.

Although these two models (i.e., NFSv4 ACLs and draft-POSIX ACLs) have some common goals and are presented in a common XDR framework, they are substantially different, for reasons listed below. Previous specifications have assumed that server implementations of the draft-POSIX ACLs could serve to support NFSv4 ACLs and ignored the issues that the semantic differences would pose for clients attempting to support the draft-POSIX model.

* The draft-POSIX ACLs address only the first of the motivations for extension, while the NFSv4 ACL model is intended to address both of them, by defining a large range of bits in the ACE mask, rather than the three POSIX bits.

* NFSv4 ACLs, by supporting DENY entries allow specific privileges to be allowed for most members of a group and be denied to some particular users.

* NFSv4 ACLs provide additional security-related facilities in addition to authorization control, through the use of AUDIT and alarm ACEs.

* These models implement very different approach to authorization in that the NFSv4 ACL model is based on the sequential scanning of all ACEs to verify authorization for all requested actions, while the draft-POSIX ACL model scans a limited subset of these and requires that all authorization for each requested action be determinable based on a single ACE.

* Both have very different approaches to managing the interaction of the mode attribute and ACL-related attributes. In particular, draft-POSIX ACLs use the middle three of the nine bits controlling authorization to establish an ACL mask applying to all ACEs except those controlling access to the owning user and to everyone.

* They have very different approaches to the management of ACL inheritance.

{Author Aside (Item #61a)]: In order to justify an eventual shift of the Acl and Dacl attributes back to be truly OPTIONAL, it is necessary to define for each file system, the type of ACL semantics provided, using information such as that provided by the ACL_Choice attribute. In so doing, we will have to make provision for various hybrids if such implementations actually exist, while not necessarily seeking to preserve the ability to generate other such potential hybrids, in all cases.

[Consensus Needed, Including List (Item #61a)]: The determination of the type of ACL semantics proceeds as follows:

* If the aclsupport attribute indicates that either AUDIT or ALARM ACEs are supported, then it can be assumed that, in general, NFSv4 ACL semantics are provided, although some OPTIONAL ACE mask might not supported.

* If the ACL_Support attribute is not supported, then if the Sacl attribute is supported then it also can be assumed, as above, that NFSv4 ACL semantics are provided.

* Otherwise, If the ACL_Support attribute is not supported then the presence of support for DENY ACEs determines whether support for NFSv4 ACL semantics is provided. However, it is required that clients determine whether support for DENY ACEs is provided by attempting to set ACLs containing such ACEs

* In the case in which neither the ACL_Support attribute nor the SACL attribute is supported, then it is not possible to determine of support for NFSv4 ACL semantics, as opposed to core UNIX ACL semantic is provided.

  As a consequence, server implementations providing support for many NFSv4 extensions need to support the ACL_Support attribute. This is because, if they do not, the client could legitimately assume that support for the NFSv4 ACL model is not present.

5.7. The Acl Attribute (v4.0)

This per-object attribute consists of an array of Access Control Entries which apply to operations performed on the current object, controlling authorization and monitoring of attempted operations.

[Consensus Needed (Item #118b)]: When this attribute is interrogated, it either returns an array of all ACEs in effect for this object including those related to authorization and to the monitoring of attempted operations. When there are no such ACEs, a zero-length array is returned, indicating that there is no ACL (and thus no SACL or DACL) associated with the object.

[Consensus Needed (Item #118b)]: When this attribute is set the values of the SACL and DACL attributes are also affected. When it is set to a zero-length array, indicating the deletion an ACL, both the SACL and DACL attributes will subsequently return an indication that there is no such ACL. If the ACL value being set only contains ALLOW and DENY ACEs, the SACL attribute will subsequently return an indication that there is no such ACL. Similarly, if the ACL value being set only contains AUDIT and ALARM ACEs, the DACL attribute will subsequently return an indication that there is no such ACL.

This attribute, consists only of an ACE array and does not support client-managed (aka automatic) inheritance). Although some inheritance features might be supported, some of the inheritance features of the draft-POSIX ACL model are not accessible using this attribute.

The acl attribute is OPTIONAL and there is no requirement that a server support it. However, when the dacl attribute is supported, it is a good idea to provide support for the acl attribute as well, in order to accommodate clients that have not been modified to use the dacl attribute.

[Consensus needed, Including List (Item #65a)]: While the original intention was to define a usable OPTIONAL attribute based on the NFSv4 ACLs defined previous specifications, it is now more appropriate to designate this under-specified attribute as experimental although still formally OPTIONAL, until the items below have been addressed.

* The intention to support, as values of this attribute two different ACL approaches, each with its own semantics. These include both the NFSv4 ACLs based on the Windows ACL model and a subset based on the more restricted semantics provided by the withdrawn POSIX ACL document with a straightforward mapping of those into the format of NFSv4 ACLs.

  The association of two such different semantic models without giving the client a way to determine which semantic model is in effect makes interoperability essentially impossible to provide.

* The potential interoperability problems are vastly expanded by the specific method by which these two models are supported.

  Instead of allowing servers to choose between these two approaches, e.g. by using the term "MAY", most statements regarding ACL semantics use the term "SHOULD", described in the text as "intentional", apparently assuming that the result is essentially equivalent to the use of "MAY". Even apart from the misuse of the terms defined in [RFC2119], this has the effect of replacing a single choice by allowing a large number of uncoordinated choices, exponentially raising the number of possibly valid semantic models that clients and users have to accommodate.

* It is not clear how far this pick-and-choose approach extends. In the case of the ace mask bits which are finer-grained than the three bits in the mode and in POSIX ACLs, there is no explicit text indicating how the coarser-grained approach would be supported by a server built to support POSIX ACLs, leaving the actual requirements uncertain.

* Although some efforts have been made to limit the damage caused by this specification uncertainty by urging clients to determine authorization decisions using ACCESS rather than by examining the ACL itself, this only addresses half of the problem and the question of what ACL to set to effect a particular authorization regime remains unaddressed, limiting the usefulness of the ACL-related features.

  Although significant efforts have been made to widen the information returned by ACCESS beyond the three-bit POSIX model, there are still cases in which it is insufficiently fine-grained. For example, adding a new file and a new sub-directory which have different ACE mask bits are both represented by a single bit in ACCESS.

[Author Aside]: Although it has generally been assumed that changes to sacl and dacl attributes are to be visible in the acl and vice versa, NFSv4.1 specification do not appear to document this fact.

[Consensus Item, Including List (Item #16a)]: For NFSv4.1 servers that support Both the acl attribute and one or more of the sacl and dacl attributes, changes to the ACE’s need to be immediately reflected in the other supported attributes:

* The result of reading the dacl attribute MUST consist of a set of ACEs that are exactly the same as the ACEs ALLOW and DENY ACEs within the acl attribute, in the same order.

* The result of reading the sacl attribute MUST consist of a set of ACEs that are exactly the same as the AUDIT and ALARM ACEs within the acl attribute, in the same order.

* The result of reading the acl attribute MUST consist of a set of ACEs that are exactly the same as the union of ACEs within the sacl and dacl attributes. Two ACEs that both appear in one of the sacl or dacl attributes will appear in the same order in the acl attribute.

5.8. The ACL_Support Attribute (v4.0)

A server need not support all of the ACE types described in Section 6.1. This attribute indicates which ACE types are supported for the current file system by any of the acl, sacl, or dacl attributes.

[Consensus Needed (Item #61b)]: Although this attribute is OPTIONAL, there are important reasons, in certain cases, to provide support, as described in Section 5.6.

The bitmask constants used to represent the abovementioned definitions within the aclsupport attribute are as follows:

```
<CODE BEGINS>
      const ACL4_SUPPORT_ALLOW_ACL    = 0x00000001;
      const ACL4_SUPPORT_DENY_ACL     = 0x00000002;
      const ACL4_SUPPORT_AUDIT_ACL    = 0x00000004;
      const ACL4_SUPPORT_ALARM_ACL    = 0x00000008;
<CODE ENDS>
```

[Author Aside (Item #14b)]: Even though support for ACL_Support is OPTIONAL, there has been no mention of the possibility of it not being supported.

[Consensus Needed (Item #14b)]: If this attribute is not supported for a server or filesystem, the client is entitled to assume that, if the acl attribute is supported, support for ALLOW ACEs is present. Thus, if such a server supports the sacl attribute, clients are not likely to use it if aclsupport is not supported by the server.

[Previous Treatment (Item #110a)]: Servers that support either the ALLOW or DENY ACE type SHOULD support both ALLOW and DENY ACE types.

[Author Aside, Including List: (Item #110a)]: The use of "SHOULD" in the preceding is unhelpful for the following reasons:

* While it is unclear what the intention is, it is certainly is not in accord with RFC2119 since there is no indication of potential harm or what might be valid reasons to do otherwise.

* While it might be one of "intentional" SHOULDs, that would make the paragraph meaningless since such SHOULDs are essentially equal to MAYs.

* The most likely source of divergence, the fact that UNIX ACLs do not support DENY ACEs, is not mentioned at all.

[Consensus Needed (Item #110a)]: Servers that support the DENY ACE type MUST support the ALLOW ACE type as well.

[Consensus Needed, Including bulleted list (Item #110a)]: Clients should not attempt to set an ACE unless the server claims support for that ACE type. The server MUST reject requests with NFS4ERR_ATTRNOTSUPP if any of the following apply:

* If the server receives a request to set an ACE type that is not allowed as part of the acl attribute being set.

* If the server receives a request to set an ACE, it cannot store.

Support for any of the ACL attributes is OPTIONAL. However, certain restrictions apply regarding the interaction of support for these attributes, A server that supports either of the newer ACL attributes (dacl or sacl) MUST support use of the new ACL attributes to access all of the ACE types that it supports. In other words, if such a server supports ALLOW or DENY ACEs and the sacl attribute, then it MUST support the dacl attribute and any ALLOW or DENY ACE types supported by the acl attribute MUST be supported in the dacl attribute as well. Similarly, if it supports AUDIT or ALARM ACEs and the dacl attribute, then it MUST support the sacl attribute any AUDIT or ALARM ACE types supported by the acl attribute MUST be supported in the dacl attribute as well.

5.9. The ACL_Choice Attribute (v4.1 extension for bis)

[Consensus Needed (Item #105f), Through end of section]:

The ACL_Choice attribute provides information regarding the server’s provision of various OPTIONAL elements or the adoption of particular valid behavioral choices in implementing NFSv4 ACLs and using them for authorization or other security-related functions. Thes include:

* The support (or not) of various OTIONAL extensions to the UNIX ACL model. These include many cases in which the optional character of these extensions was not exlicitly recognized in previous specifications.

* Many cases in which the proper server behavior was not made clear due to the sort of underspecification discussed in Section 3.3.

* [Consensus needed (Item #121b, #122b, #123b, #124b)]: The need to incorporate features necessary to fully support Windows ACLs and to use them in a POSIX-supportng context in a way consonant with the POSIX approach to security.

This has been made necessary by the approach taken in previous specifications, in which numerous extensions of the core UNIX ACL model are mentioned as possibilities but there is no attention at all to the problems that this creates for clients without a way to determine whether any particular extension is supported. Without such facilities, the inability to determine whether the necessary extensions are supported makes it difficult for clients to use the extensions provided. We consider that inability as a protocol defect justifying the extension of the protocol via the addition of a new OPTIONAL attribute.

While it might be supposed that this attribute could be rendered unnecessary since the server would be expected to reject requests for unsupported services, the definition of ACLs in previous specifications make that impossible for the following reasons:

* In many cases, existing specifications are written, so that such rejection is not required and sometimes it is suggested that it is undesirable. For example, [RFC8881] contains the following.

  > It is tempting to accomplish this by rejecting any ACL that falls outside the small set that can be represented accurately. However, such an approach can render ACLs unusable without special client-side knowledge of the server’s mapping, which defeats the purpose of having a common NFSv4 ACL protocol. Therefore, servers should accept every ACL that they can without compromising security.
  >
  > So it may be useful for a server to accept an ACL even if not all of its modules are able to support it.
  >
  > For example, suppose a client tries to set an ACE with ACE4_FILE_INHERIT_ACE set but not ACE4_DIRECTORY_INHERIT_ACE. If the server does not support any form of ACL inheritance, the server should reject the request with NFS4ERR_ATTRNOTSUPP. If

the server supports a single "inherit ACE" flag that applies to both files and directories, the server may reject the request (i.e., requiring the client to set both the file and directory inheritance flags). The server may also accept the request and silently turn on the ACE4_DIRECTORY_INHERIT_ACE flag.

As a result of this and similar statements in the existing specifications, it is impossible to rely on rejection in the cases we concerned with.

* Much of the existing vagueness about the handling of ACE mask bits is exacerbated by the false assumption that all such mask bits can be treated as part of granularity hierarchy mappable to the coarse-grained approach of three permissions derived from POSIX.

* A further source of difficulty arises from the possibility, alluded to in previous specifications that a server might accept ACLs it is not prepared to enforce, in the expectation that another that is capable of enforcing the ACL might later do so.

  As a result, ACLs could be accepted with this expectation, with no way of determine whether they are enforced where first created or any other server to which they might be copied.

While it is now easy to see that the choices above were ill-advised, it is impossible to change them now without created compatibility issues not acceptable in the bis context. As a result, our only realistic option is to define a protocol extension to correct this defect as provided for by Section 9 of [RFC8178]. The structure of this new ACL_Choice attribute as described in Section 5.9.1

5.9.1. Structure of the ACL Attribute

```
<CODE BEGINS>
const FATTR4_ACL_Choice     = 87; -

struct nfs4acdi {
            uint32_t    acd_offset;
            uint32_t    acd_length;
};

struct nfs4acas {
            bitmap4     aca_bits;
            nfs4acdi    aca_ditems<>;
            opaque      aca_dpool<>;
};

typedef nfs4acas fattr4_ACL_Choice;
<CODE ENDS>
```

The ACL_Choice attribute, like other NFSv4 features such as attribute set packaging and the provision of pNFS layout-type-specific data, avoids some of the limitation of the typical definition of XDR structure by defining a nominally opaque arrays and providing means by which the actual structure of the formally opaque array, in this case aca_dpool, is described by other data items which allow various sections of the opaque array to be conveniently accessed by considering some parts of it as overlaid by other XDR data structures.

The nfs4_acas which describes the attribute consists of the following items:

* aca_bits is a bitmap containing flags about implementation choices. The bit numbers are defined in Section 5.9.2.

* aca_ditems is an array indexed by the data item indices defined a value with the enum acc4dinums, defined in Section 5.9.3.

  Each nfs4acdi within aca_ditems specifies where, within aca_dpool the data .in the form indicated by the appropriate row within Table 3, is located. When acd_length is zero, there is no corresponding data item in aca_dpool.

* The aca_dpool fields contain an array of data used to provide space for the items whose location in indicated within aca_ditems.

5.9.2. ACL_Choice Flag Bit Numbers

[Consensus Needed (Items #123c, #124c), Through end of section]:

```
<CODE BEGINS>
enum acc4bitnums {
            ACC4BN_NEINGM   = 0,
            ACC4BN_SEPFWX   = 1,
            ACC4BN_SEPAFD   = 2,
            ACC4BN_SEPDE    = 3,
            ACC4BN_RNASDI   = 4,
            ACC4BN_NAD      = 5,
            ACC4BN_NADMOD   = 6,
            ACC4BN_MBCA     = 7,
            ACC4BN_SMJUST3  = 8,
            ACC4BN_SMOLD    = 9,
            ACC4BN_SMFULL   = 10,
            ACC4BN_3MASKB   = 11,
            ACC4BN_AUTHWHO  = 12,
            ACC4BN_IN1BIT   = 13,
            ACC4BN_INHFULL  = 14,
            ACC4BN_INHAUTO  = 15,
            ACC4BN_RVINV    = 16,
            ACC4BN_OTHWHO   = 17,
            ACC4BN_SVTX     = 18,
            ACC4BN_SVTXOLD  = 19,
            ACC4BN_CRWHO        = 20,
            ACC4BN_ORIGHTSWHO   = 21
};
<CODE ENDS>
```

The tables below provide information about the individual implementations that are communicated in the ACL_Choice attribute. One table is used for specific bits for cases in which the choice is specified with the associate bit map. The other is used when the choice is combined with some additional info in a subsidiary data item. In both cases, the table identifies the choice class while is selected from the following list:

Mbcs: Mask bit with clear semantics

Mbus: Mask bit with uncertain semantics

Obv: Original behavioral variant

Lbv: Later behavioral variant

These tables and the sections they point to will contain restrictions making certain sets of flags or data items invalid. When such restrictions are violated, the client is free to ignore the ACL_Choice attribute and treat it as unsupported.

Id	Class	Section	Description
BN_NEINGM	Obv	S 6.6	Whether ALLOW ACEs for named entities contribute to the group mode bits.
BN_SEPWX	Mbcs	S 6.4	Whether to separately authorize extension of a file and rewriting of existing bytes.
BN_SEPAFD	Mbcs	S 6.4	Whether to separately authorize adding of subdirectories and additional of other sort of directory entries.
BN_SEPDE	Mbcs	S 6.4	Whether to separately authorize deletion of directory entries and other modifications of directories.
BN_RNASDI	Lbv	S 6.6	Whether Rename within a directory are to require specific permissions to add and delete entries within the encompassing directory.
BN_NAD	Mbus	S 6.4	Whether to separately authorize access to and modification of named attribute directories
BN_NADMOD	Lbv	S 6.6	Whether to use the mask bits associated with named attributes to control operations within those directories, in addition to the use of OPENATTR.
BN_MBCA	Obv	S 6.6	Whether the server is allowed to convert sets of mask bits or other flags it cannot enforce or undertake to store and retain unmodified to mask bits it does support, rather than returning an error.

BN_SMJUST3	Obv	S 6.6	Whether the server when processing a change to the mode, sets the acl to represent the handling specified by those mode bits, making no effort to preserve other aspects of the ACL Although this appears to be an alternative to BN_SMOLD and BN_SMFULL, there are situation in which multiple of these can be true simultaneously
BN_SMOLD	Obv	S 6.6	Whether the server when processing a change to the mode, sets the acl to represent the handling specified by those mode bits, while making the efforts to preserve other aspects of the ACL as recommended by the existing specifications. Although this appears to be an alternative to BN_SMJUST3 and BN_SMFULL, there are situation in which multiple of these can be true simultaneously
BN_SMFULL	Obv	S 6.6	Whether the server when processing a change to the mode, sets the acl to represent the handling specified those mode bits, while making the efforts to preserve other aspects of the ACL as recommended by Section 10.7 Although this appears to be an alternative to BN_SMJUST3 and BN_SMOLD, there are situations in which multiple of these can be true simultaneously.
BN_3MASKB	Mbus	S 6.4	Indicates the server only

			supports the three ACE mask bits derived directly from the POSIX privilege bits.
BN_AUTHWHO	Obv	S 6.6	[Consensus Needed (Item #50a), Entire entry]: Indicates whether the server treats ACEs with the special who values related to authentication defined in Section 8.4.3 as valid. Although existing specifications treat this as a feature, the uncertainty associated semantics makes it inappropriate to treat it that way so we will treat the acceptance of ACEs containing such values of the "who" field as behavioral variants.
BN_INNO	Obv	S 6.6	Indicates there is no support for ACL inheritance.
BN_IN1BIT	Obv	S 6.4	Indicates there is some support for ACL inheritance but no support for separate inheritance bits for file and directories
BN_INHFULL	Mbcs	S 6.4	Indicates there is full support for ACL inheritance but not necessarily for client-managed (aka automatic) inheritance.
BN_INHAUTO	Mbcs	S 6.4	Indicates that there is support for Windows automatic ACL inheritance, which is client-managed.
BN_RVINV	Lbv	S 6.6	Indicates that the server will, in some cases, when presented with incorrect or unsupported, ACLs modify them to be valid/acceptable rather

			than rejecting them.
BN_OTHWHO	Obv	S 6.6	[Consensus Needed (Item #50a), Entire entry]: Indicates whether the server treats ACEs with the special who values not related to either authentication or support for draft-POSIX ACL semantics that are defined in Section 8.4.3 as valid. Although existing specifications treat this as a feature, the absence of any specification of associated semantics makes it inappropriate to treat it that way so we will treat the acceptance of ACEs containing such values of the "who" field as behavioral variants.
BN_SVTX	Lbv	S 6.6	[Consensus Needed (Item #12a), Entire entry]: Indicates that the server, in authorizing file deletion and RENAME operations, takes special account of the setting of MODE4_SVTX in the mode attribute of the encompassing directory. This bit, like BN_SVTXOLD, applies to POSIX-based as well as to ACL-based authorization. This is necessary since the two have to work together, irrespective of whether various entities involved have ACLs. When this bit is set, ACL-based authorization semantics reflects the treatment set out in Section 8.3.20. This bit

			should not be set together with BN_SVTXOLD. When authorization of deletes and RENAMEs depends on the set of MODE_SVTX and ACLs are not supported, in is desirable that ACL_Choice be supported nevertheless, with this bit set to indicate the use of MODE4_SVTX in authorization.
BN_SVTXOLD	Lbv	S 6.6	[Consensus Needed (Item #12a), Entire entry]: Indicates that the server, in authorizing file deletion and RENAME operations, takes special account of the setting of MODE4_SVTX in the mode attribute of the encompassing directory. This bit, like BN_SVTX, applies to POSIX-based as well as to ACL-based authorization. This is necessary since two have to work together, irrespective of whether various entities involved have ACLs. When this bit is set, ACL-based authorization semantics reflects the treatment set out in Section 8.3.20. This bit should not be set together with BN_SVTX.
BN_CRWHO	S 6.4	Mbcs	Indicates that the server supports the special creator-based "who" values CREATOR_OWNER@ and CREATOR_GROUP@.
BN_ORIGHTSWHO	S 6.4	Mbcs	Indicates that the server supports the special creator-based "who" value

			OWNER_RIGHTS@.

Table 2: ACL_Choice flag bits

5.9.3. ACL_Choice Data Items

```
<CODE BEGINS>
typedef uint32_t        ombr4word;

enum acc4dinums {
            ACC4IN_INFMB    = 0,
            ACC4IN_ODDMB    = 1,
            ACC4IN_STOREUA  = 2,
            ACC4IN_TSUPP    = 3,
            ACC4IN_WINSA    = 4,
            ACC4IN_ASLIM    = 5
};
<CODE ENDS>
```

Words of type ombr4word are used to describe the particular usage of ace mask bits whose semantics is not currently determinable. The definition of the individual bit fields appears in Section 8.3.13.

The individual data items that can be included are described in the table below.

Id	Class	Ov. Sect.	Det. Sect.	Data	Description
IN_INFMB	Fwsr	S 6.5	S 8.3.13	ace4maskinfo	Provides information relating to the support of ACE mask bits whose scope can be accounted for as either one of the core mask bits or as a finer-granularity variant of file ownership or one of the core mask bits
IN_ODDMB	Fwsr	S 6.5	S	ombr4word<>	Provides

			8.3.13		information relating to the support of ACE mask bits defined within Section 8.3.10, whose scope cannot accounted for as a finer-granularity variant of one of the three POSIX privilege bits or file ownership Also used to provide information about mask bits defined in Sections 8.3.6, 8.3.8, and 8.3.9, when the treatment of that mask bit does not fully accord with the requirement of the defining section.
IN_STOREUE	Fwsr	S 6.5	S 12.2	acc4storeua	Provides information relating to the support of ACE mask bits, ACE type, and special who values which are not enforced by current NFSv4 server implementation but are stored and retrieved so that ACLs with these feature can be preserved for use by other NFSv4 servers, other remote file

					access protocol, or for local file access.
IN_TSUPP	Fwsr	S 6.5	S 8.1.2	uint32_	Provides information relating to the support of ACE types, including newer ones not included in the ACL_Support attribute.
IN_WINSA	Fwsr	S 6.5	S 12.3	acc4_winsa	Provides information relating to the support of ACE mask bits meant to support Windows-based semantics that might be affected by special accommodations to provide support for protocols such as NFSv4[01] for which such semantic support is unavailable.
IN_ASLIM	Fwsr	S 5.5	S 12.4	acc4_aslim	Provides filesystem-based limits on the size of ACLs t be presented for storing or verify-based comparison or to be fetched by the client.

Table 3

5.10. The Dacl Attribute (v4.1)

[Consensus needed (Item #127a), through end of section]: The dacl attribute was defined in order to divide ACLs so that the authorization-related entries (i.e. ALLOW and DENY entries) were no longer combined in the same attribute as AUDIT and ALARM entries. In addition, this attribute includes support for "automatic" inheritance as decribed in Section 8.2.3.

[Consensus Needed (Item #118c)]: When this attribute is interrogated, it returns an array of ACEs of types ALLOW and DENY, in an order consistent with the ordering of those ACEs in the Acl attribute. When there are no such ACEs, it return a zero-length ACE array, indicating the absence of a DACL associated with the current file object.

[Consensus Needed (Item #118c)]: When this attribute is set to a value represent by a zero-length ACE array, all ALLOW and DENY ACEs are removed from the Acl attribute. If there were no AUDIT and ALARM ACEs, the Acl attribute will subsequently return a zero-length ACE array indicating the absence of an ACL for the current file object.

[Consensus Needed (Item #118c)]: When this attribute is any other value, any existing ALLOW and DENY ACEs are removed from the Acl attribute and the ones specified are added. Subsequently the Acl and Dacl attributes will not return a zero-length array because there will be an ACL an a DACL associated with the current file object.

{Consensus needed, Thru rest of Section (Item #65b)]: While the original intention was to define a usable OPTIONAL attribute based on the NFSv4 ACLs defined previous specifications, it is now more appropriate to designate this under-specified attribute as experimental although still formally OPTIONAL until the issues discussed in Section 5.7 are addressed

Although the issues applying to the acl attribute apply equally to the dacl attribute, given the description in earlier specifications, it might be easier to resolve them in the case of the dacl attribute for the following reasons:

* Implementations of POSIX ACLs might not have been updated to support the sacl attribute, since doing so would add no value.

* Even if such POSIX-ACL-oriented implementations of the sacl attribute did exist, it might be easier to get agreement on regularizing the sacl attribute since, if acl were left as it is, the POSIX ACL support would still be available.

5.11. The Sacl Attribute (v4.1)

[Consensus needed (Item #127b), through end of section]: The sacl attribute is similar to the acl attribute, but sacl only allows the inclusion of AUDIT and ALARM ACEs. The sacl attribute supports automatic inheritance and the marking of ACEs using the ACe4_INHERITED_ACE flag (see Section 8.2.3).

[Consensus Needed (Item #118d)]: When this attribute is interrogated, it returns an array of ACEs of types ALLOW and DENY, in an order consistent with the ordering of those ACEs in the Acl attribute. When there are no such ACEs, it return a zero-length ACE array, indicating the absence of a DACL associate with the current file object.

[Consensus Needed (Item #118d)]: When this attribute is set to a value represent by a zero-length ACE array, all AUDIT and ALARM ACEs are removed from the Acl attribute. If there were no ALLOW or DENY and ACEs, the Acl attribute will subsequently return a zero-length ACE array indicating the absence of an ACL for the current file object.

[Consensus Needed (Item #118d)]: When this attribute is any other value, any existing AUDIT and ALARM ACEs are removed from the Acl attribute and the ones specified are added. Subsequently the Acl and sacl attributes will not return a zero-length array because there will be an ACL an a SACL associated withe current file object.

{Consensus needed, Thru rest of Section (Item #65c)]: While the original intention was to define a usable OPTIONAL attribute based on the NFSv4 ACLs defined in previous specifications, it is now more appropriate to designate this under-specified attribute as experimental although still formally OPTIONAL until the issues discussed in Section 5.7 are addressed

The Sacl attribute was added in NFSv4.1 in order to divide ACLs so that the non-authorization-related entries (i.e. AUDIT and ALARM entries) would no longer be combined in the same attribute with the ALLOW and DENY entries.

[Author Aside, Including List (Items #61c, #105g, #110b)]: Although the existing discussion of ACE structure results in the same sort of lack of clarity affecting the Acl and Dacl attributes, it us more likely that these will resolved in the case of the Sacl attribute as compared to the Acl or Dacl attributes, even though the problems with the existing text are essentially the same.

* There are no AUDIT or ALARM entries, in POSIX ACLs, so there would be no need accommodate existing implementations of these that embody a more POSIX-oriented semantic model.

  As a result, it is likely to be easier to get WG approval for changes that clearly state that the ACE mask bits are to followed strictly for the these types of ACEs.

* Since such entries have no role in compute a corresponding mode attribute, the effect of this issue for the sacl attribute is not problematic.

6. Structure of the ACL-related Features

[Consensus Needed (Items #104g, #105h), Through end of section]:

Until now, the set of OPTIONAL features, had been limited to the set of ACL-related attributes: acl, ACL_Support, sacl, and dacl. Many of these had a vast semantic range which resulted from the existence of two different ACL models and the way in which previous specifications attempted to accommodate those two models and the range of behaviors provided by existing file systems.

In this specification, we will take a different approach in which, where multiple approaches are to be allowed to server implementations, we explicitly designate them as OPTIONAL and make them visible to the client using the ACL_Support and ACL_Choice attributes.

These choices had a number of sources and were previously dealt with in a number of ways:

* Choices that arose from differences in bit mask granularity arising from server choices as to which mask bits to support are dealt with by providing for each such finer-grained mask bit a presumed mask bit ancestor and organizing the mask definitions on that basis.

* Other cases in which multiple behavioral variants were previously allowed are treated as OPTIONAL features, as described below.

  This includes many cases in which the allowance of multiple behavioral variants seems ill-advised, arising from the issues discussed in Section 3.3.

An important set of choices arose from the expansion of the set of distinct actions that could be subject to authorization control using ACLs. The options ranged from no expansion relative to the three

privilege bits inherited from POSIX to the major expansion of the set of ACE mask bits in [RFC3530],drived from embracina Windows-oriented granlarity for authorization.

Each of these additions is best thought of as a distinct feature. Most of them, for which the basic semantics can be inferred, based on existing specifications, are discussed in Section 6.4.

In previous specifications, it was assumed that the ACE mask bits other than those corresponding to three POSIX permission bits could all be addressed as finer-grained variants of one of those permission bits. This is now known not to be the case because:

* The role of file ownership in establishing authorization for certain actions was ignored.

* The relationship of the X-bit and R-bit and their role in providing authorization was not taken account of.

* The authorization for many actions is uncertain or depends, invarious existing implementations, on a combination of permission bits and ownership.

A number of cases in which multiple behaviors are allowed will be addressed by defining the choice as a feature even without any clear benefit to either approach. These are discussed in Section 6.6.

These include cases in which multiple behaviors were allowed to accommodate existing server behavior and also some cases in which the utility of the approach was doubtful, and a different approach was specified.

Because of the development of many distinct server implementations with a specification that took the semantically lax approach discussed above, there will inevitably be more such cases as work on ACLs proceeds. Because of this, agreement on a uniform approach to such issues is unlikely at this point, so the ACL_Choice attribute might need to extended to accommodate them as described in Section 12.1

6.1. Role of the Core UNIX ACL model

[Consensus Needed (Items #64a, #105i), Through end of section]

Although the working group did not adopt the ACLs in the withdrawn POSIX draft, the continued existence of implementations of them and variants sharing important features with them in non-NFSv4 UNIX contexts has created protocol difficulties that need to be resolved.

In many cases, such ACLs and their associated semantics are the basis for ACL support in UNIX client-side APIs and in UNIX file systems supported by NFSv4.

In the context of this document, the Core UNIX ACL model is defined as any approach which shares all of the following features:

* Supports only the ACE mask bits that are direct replacements for the three POSIX privilege bits.

  When the ACL_Choice attribute is supported, this characteristic is reported to the server using the flag bit ACC4BN_3MASKB.

* Supports only ALLOW ACEs.

* [Consensus Needed (Item #117h)]: Has no support for the NFSv4 approach to ACL inheritance, although there can be support for default ACLs. For a discussion of how this affects support for handling of POSIX ACLs, see Section 6.2.

The following characteristics of draft-POSIX ACLs are not included in this definition but are treated as allowable behavioral variants within the UNIX ACL model:

* The inclusion of mask bits derived from ALLOW ACEs for named users and groups in the group mode bits.

  When the ACL_Choice attribute is supported, this characteristic is reported to the server using the flag bit ACC4BN_NEINGM.

* The non-support for partial satisfaction of ALLOW ACEs, so that each authorization request can only be supported by a single ALLOW ACE.

  When the ACL_Choice attribute is supported, this characteristic is reported to the server using the flag bit ACC4BN_AANPS.

[Consensus Needed (Item #117h)]: Although the semantic range of core UNIX ACLs is a subset of that for NFSv4 ACLs, expecting clients to perform the mapping between NFSv4 ACLs and draft-POSIX ACL on their own has not worked well. Those issues are addressed for NFv4.1 as described in Section 6.2 while possible approaches to correct this situation are discusssed in Appendix E.

6.2. Potential Support for the POSIX ACL Model

[Consensus Needed (Item #117i), Though end of section]:

A helpful summary of the draft-POSIX ACL model can be found in [Gr端nbacher]. Although this document deals only with the Linux implementation, the guidance it provides regarding required semantics is expected to be helpful to other potential implementations as well.

Although considerable efforts were made to accommodate POSIX ACL implementations with in the existing NFSv4 ACL model, those efforts left support for POSIX ACLs in an unfortunate state for the following reasons:

* Many important parts of the POSIX ACL model, such as inheritance were not REQUIRED and no attention was paid to the differences between those functions in the two different ACL models that were to be supported.

* While efforts were made to accommodate semantic differences between those models, the effect was limited since many differences were simply ignored due to a lack of interest in semantic description. In addition, behavioral differences that were recognized were addressed in a way that gave the client no ability to select its preferred variant or even determine the choice made by the server.

The development of the ACL_Choice attribute served to address some of these issues, but did not provide full support, as discussed below.

* The uncertainty about the mapping between ACL and modes (and between core UNIX ACLs an server NFSv4 ACLs has been addressed by making this server choice one that the client can find out about, where ACL_Choice is supported.

  Fully addressing this variability would most likely require a significant degree of attribute incompatibility making it only possible in NFSv4.2 or later.

* The differences in the processing of ALLOW ACEs has been addressed by making either of the approaches an allowable server behavioral variant as was done in the mode computation case discussed above. As part of that change, the server choice was made one that the client can find out about, where ACL_Choice Adding more discussion of Windows semantic requirements to the Introduction (Section 1) to reflect the need to clearly describe these features despite the fact they are no longer considered the core of the new ACL model. is supported.

The fact that this had not been done previously was treated as an easily remedied oversight, given the clear intention to allow servers supporting POSIX ACLs to be used, without, as previous specifications put it, "invalidating" them.

As a result, it became necessary to accommodate support for draft-POSIX ACLs as a separate ACL model. The details of that model are best deferred to an extension of a later minor version as discussed in Appendix E. What is necessary in this document is to appropriately prepare for future specification of multiple ACL models by doing the following things:

* Distinguishing between ACLs in general and the specifics of NFSv4 ACLs, that, for some time, are expected to be the only ACL structures that exist. Since it is difficult to name NFSv4 ACL structure as anything other than "ACL", we retain that naming but refer to the general concept as "Access Control List". In addition, the term "Authorization Control List" will refer to any ort of Access Control List that is only capable of controlling authorization, such as the Dacl attribute.

* Distinguishing between ACEs in general and the specifics of NFSv4 ACEs, that, for some time, are expected to be the only ACE structures that exist. Since it is difficult to name NFSv4 ACE structures as anything other than "ACE", we retain that naming but refer to the general concept as "Access Control Element".

* Making it clear that the material in Section 10 applies only to NFSv4 ACLs and that the handling of coordination of the mode with other sorts of Access Control Lists is the jobs of the extensions defining those new models.

6.3. Server Behavioral Restrictions to Apply when ACL_Choice is not Supported

[Author Aside (Item #116a)]: This section explores possible additional behavioral restrictions to avoid situations in which the lax approach to the specification of server semantics could unacceptably interfere with client operation. Given the difficulty of supporting NFSv4 extensions without the use of ACL_Choice, we are focusing for now on NFSv4-oriented clients, although the working group might consider what could be done for other clients in later drafts.

[Consensus Needed (Items #116a, #121c), Through end of section]:

Because of the uncertainty about the proper handling of the ACE mask bits defined in Section 8.3.10, the recommendations below SHOULD be adhered to by servers that do not support the ACL_Choice attribute in order to allow clients to convert UNIX ACLs to NFSv4 ACLs as described in Section 8.3.19. In the context of the above recommendations, the only valid reason to bypass them is reliance on previous specifications that do not make clear the importance of adhering to these constraints. Further, those implementing such servers need to be aware that they are compromising the utility of the implementation for UNIX-oriented clients and that other clients might not be able to with their semantics either.

When transforming an ALLOW ACE using only the three ACE mask bit that correspond to POSIX privilege bits, the corresponding bit mask SHOULD be constructed as follows:

* When ACE4_READ_NAMED_ATTRS is not excluded (i.e., when named attributes are supported) it is to be set if the mask bit corresponding to the read privilege bit (i.e., ACE4_READ_DATA or ACE4_LIST_DIRECTORY) is set.

  The bit will also be set in other circumstances, such as when the ACE4_EXECUTE MASK bit is set for a non-directory object.

* When ACE4_READ_EXT_ATTRS is not excluded (i.e., when authorization of extended attributes operation is supported) it is to be set if the mask bit corresponding to the read privilege bit (i.e., ACE4_READ_DATA or ACE4_LIST_DIRECTORY) is set.

  The bit will also be set in other circumstances, such as when the ACE4_EXECUTE MASK bit is set for a non-directory object.

* ACE4_READ_ATTRIBUTES is always to be set.

* ACE4_READ_ACL is always to be set.

* [Consensus Needed (Item #62a), Through end of bulleted item]: ACE4_DELETE is always to be set.

  This applies to authorization in the case in which the "sticky" bit for the directory is set as well as the case in which it is not set.

* ACE4_WRITE_RETENTION and ACE4_WRITE_RETENTION_HOLD are not to set unless the mask bit corresponding to the write privilege bit (i.e., ACE4_WRITE_DATA or ACE4_ADD_FILE) is set.

Servers MAY condition these settings on other conditions (e.g., only setting the bit for ACEs for the owner of a file).

The above recommendations do not apply when ACL_Choice is supported because the information provided allows the mapping described in Section 8.3.19 to be performed as needed by the semantics provided by the server.

[Author Aside (Item #116a)]: The possibility of extending these restrictions to case in ACL_Choice is supported needs to be considered, as does the possibility of follow-on restrictions regarding the details of the mapping between modes and ACLs. Much will depend on the range of existing server behaviors. This might enable further simplification of the ACC4IN_ODDMB data item.

6.4. Feature Extensions With Clear Semantics

[Consensus Needed (Items #104h, #105j), Through end of section]:

The following extensions of UNIX ACL model are now considered OPTIONAL feature whose support, on each given file system, can be determined using the ACL_Support and ACL_Choice attributes. These include some for which the core semantics is clear but there are potential behavioral differences, that resolved based on the existence of other features described in Section 6.6.

* The provision of separate authorization control for extending a file and for overwriting existing bytes is represented by the flag bit ACC4BN_SEPFWX of the ACL_Choice attribute.

  When this feature is available and different authorization apply to a file authorization decision cannot be made at OPEN time but are reflected in the results returned to ACCESS and enforced on each Write.

* The provision of separate authorization control for adding a new entry to a directory and making sorts of modification for the directory is represented by the flag bit ACC4BN_SEPAE of the ACL_Choice attribute.

  The detailed semantics of this feature are affected by the feature represented by the ACC4BN_RNASDI flag bit, discussed in Section 6.6. Note that when this bit is not set, renaming of objects within the directory would be affected by restrictions on adding new entries.

* The provision of separate authorization control for adding a sub-directories to a directory and for adding other sorts of entries to a directory represented by the flag bit ACC4BN_SEPAFD of the ACL_Choice attribute.

  This feature bit is only meaningful if feature represented by the ACC4BN_SEPAE flag bit is also in effect.

  The detailed semantics of this feature are affected by the feature represented by the ACC4BN_RNASDI flag bit, discussed in Section 6.6. Note that when this bit is not set, the ability to rename a subdirectory and a file within a directory would be controlled by different ACE mask bits and that renaming of objects within the directory would be affected by restrictions on adding new entries.

* The provision of separate authorization control for deleting a new entry to a directory and making other sorts of modification to the directory is represented by the flag bit ACC4BN_SEPDE of the ACL_Choice attribute.

  The detailed semantics of this feature are affected by the feature represented by the ACC4BN_RNASDI flag bit, discussed in Section 6.6. Note that when this bit is not set, renaming of objects within the directory would be affected by restrictions on deleting existing entries.

* The provision of separate authorization control for operations involving a file object’s named attribute directory is represented by flag bit ACC4BN_NAD of the ACL_Choice attribute.

  The detailed semantics of this feature are affected by the feature represented by the ACC4BN_NADMOD flag bit, discussed in Section 6.6. Note that when this bit is not set the creation of a new name attribute directory would be controlled by a separate ACE mask bit but this same bit would not control the authorization of otherwise modifying the named attribute directory.

* The server’s support for a limited set of mask bit derived from the three POSIX privilege bits is represented by flag bit ACC4BN_3MASKB of the ACL_Choice attribute.

  This bit cannot be validly combined with any of the bits ACC4BN_SEPFWX, ACC4BN_SEPAE, ACC4BN_SEPDFD, ACC4BN_SEPDE, or with any use of the data items ACC4IN_INFMB or ACC4IN_ODDMB.

* The server’s support for DENY ACEs is indicated by the bit ACL4_SUPPORT_DENY_ACL in the ACL_Support attribute or by the bit with numeric value two in the word associated with data item ACC4IN_TSUPP.

* The server’s support for AUDIT ACEs is indicated by the bit ACL4_SUPPORT_AUDIT_ACL in the ACL_Support attribute or by the bit with numeric value four in the word associated with data item ACC4IN_TSUPP.

* The server’s support for ALARM ACEs is indicated by the bit ACL4_SUPPORT_ALARM_ACL in the ACL_Support attribute or by the bit with numeric value eight in the word associated with data item ACC4IN_TSUPP.

* The server’s support for newly defines ACE types is indicated a bit the word associated with data item ACC4IN_TSUPP. The selected bit is one left-shifted by the value of the ACE type.

* The server support’s support for a limited form of ACL inheritance is indicated by the flag bit ACC4BN_IN1BIT. In this form of inheritance there is no provision for separate control of inheritance for subdirectories and for non-directory objects.

  Because this level of inheritance support is needed for implementation of the draft-POSIX ACL model, clients needing that support requires a file system with either ACC4BN_IN1BIT or ACC4BN_INHFULL set.

  When this form of inheritance is supported, ACEs in ACE4_FILE_INHERIT_ACE and ACE4_DIRECTORY_INHERIT_ACE are set to different values are generally rejected. However, when ACC4BN_RVINV is set the server is allowed to convert the ACE to one in which the two values are the same.

  This bit is incompatible with ACC4BN_INNO and ACC4BN_INHFULL.

* The server support’s support for full ACL inheritance is indicated by the flag bit ACC4BN_INHFULL. In this form of inheritance there IS separate control of inheritance for subdirectories and for non-directory objects.

  This bit is incompatible with ACC4BN_INNO and ACC4BN_IN1BIT.

* The server’s support for automatic client-managed ACL inheritance is indicated by the flag bit ACC4BN_INHAUTO.

  This bit in incompatible with ACC4BN_INNNO.

6.5. Features with wide Semantic Ranges

[Consensus Needed (Item #105k), Through end of section]:

There are a number of features in which the wide latitude given to server implementers creates a set of uncertainties that cannot be reduced to a small set of individual binary choices. These cases are dealt within the ACL_Choice attribute by defining a data item to provide the necessary data, as listed below:

* The ACL_Choice data item with index ACC4IN_TSUPP is used to provide information the ACE types that a server accepts and enforces. Unlike the ACL_Support attribute it is not limited to the current set of known ACE types, but is designed so that new types, added via the protocol extension mechanism have ready-made means to indicate whether support is present.

* The ACL_Choice data item with index ACC4IN_ODDMB is used to provide information about the nature of the support for ACE mask bits which are neither REQUIRED nor a finer-grained corelate of another REQUIRED mask bit.

  There are many aspects of the support for these mask bits that are not clearly discussed in the existing specifications. That is most likely because it was incorrectly assumed that all the mask bits did fit in this model, making detailed explanation necessary.

  As a result, it remains unclear, for each of these bits, how authorization for the corresponding operation is to be determined, when ACLs are generated based on the setting of mode. Similarly, their contribution, if any, to modes computed from existing ACLs needs to be clarified.

  See Section 8.3.13 for details about how such information is made available to the client, when the server supports any of these ACE mask bits.

* The ACL_Choice data item with index ACC4IN_STOREUA is used to provide information about cases in which a server accepts, store, and returns ACLs that it is not prepared to enforce.

  This includes cases in which the client specifies different handling for two mask bits that the server is not prepared to distinguish, an also cases in which there are other features of the NFSv4 ACL model that the server is not prepared to support (e.g. DENY ACEs, ACL inheritance).

While the possibility of servers accepting ACLs that they are not prepared to enforce is mentioned in a number of places in the existing specifications, there is no normative text regarding this practice. In this document, we treat it as an OPTIONAL feature with details of the set of ACLs supported in this way described using the structures defined in Section 12.2.

Within the existing specifications, the possibility of accepting ACLs that cannot be enforced has the unfortunate consequence that clients needing particular ACL functionality might set ACLs using that functionality and have them accepted with no way of finding that the required functionality is not available.

When the ACL_Choice attribute is supported, there is a way to find out the ACE types for which support is available. Nevertheless, it is still helpful to be notified when unsupported ACLs are set. When the ACC4IN_STOREUA is not present, clients can often be assured of such notification. See Section 12.2 for details.

The handling of the ACE mask bit ACE4_SYNCHRONIZE could arguably dealt with as part of either of the above data items, since it does not fit in the varying-granularity model and, by its nature, it cannot be enforced within NFSv4 semantics. Within the context of this specification, this mask bit’s handling is addressed in the ACC4IN_ODDMB data item.

It is possible that the above data items might be eventually replaced by a small set of feature bits, as described in Section 12.1. For that to be possible, the working group would need information about servers that supported this feature and decide if a small set of bits could adequately represent the needed semantic range. That could be done for either of the above data items as part of the rfc8881bis effort, as described in Section 6.7.

6.6. Implementation Behavioral Choices

[Consensus Needed (Items #12b, #105l), Through end of section]:

There are a number of cases in which for various reasons, multiple behaviors are allowed in implementing some function. Since it is necessary that clients be aware of these differences, they are being treated as features within the ACL_Choice attribute, even though they might not be considered features in other contexts.

Such behavioral choices can arise as a result of an explicit decision to allow multiple server behaviors, or because the lack of a clear semantic description allowed multiple approaches to develop. In the latter case, we may currently be unaware of these different approaches, so that it is expected, as described in Section 12.1 to extend ACL_Choice as these different behaviors are discovered.

One additional source of such possible behavioral differences results from cases in which existing specifications describe the function of various ACE mask bits in a way that does not seem to make sense and other approaches have had to be added to allow useful authorization control. In such cases, the newer approach is presented as a distinguishable feature, at least until such time as a consensus can be arrived at to standardize on a single approach.

The following behavioral differences are treated as features within the ACL_Choice attribute:

* The ACL_Choice flag bit BN_NEINGM specifies the behavior implemented by the server in computing a mode based on an existing ACL. Because two methods are explicitly allowed in existing specifications, we represent this choice as a feature despite the lack of a clear benefit to either approach.

  When this flags is set, ALLOW ACEs for named entities contribute to the privilege bits used for member of the owning group.

  The inclusion of this choice within the ACL_Choice does not imply that each of these choices is equally desirable. It is hard to determine whether the existing specifications contains a clear recommendation regarding this choice. In any case, this is a matter that would need to be addressed in Section 12.5.

* The ACL_Choice flag bit ACC4BN_RNASDI specifies that the authorization of within-directory renames is to be controlled by permission to write the specified directory and does not requires, when these operations are authorized separately, specific permission to delete directory entries and to add directory entries of the type of object being renamed

This difference was not recognized in the existing specifications and the existing text in those documents suggests a different approach. Nevertheless, we take a different approach in this document while allowing the older approach because of the unfortunate consequences of the earlier approach for clients and server that do support a finer-grained approach to the authorization of directory-modifications. See Section 12.5 where this issue is addressed as part of Consensus Item #9, for specifics.

* The ACL_Choice flag bit ACC4BN_NADMOD specifies that authorization of operations accessing or modifying named attribute directories is to be controlled as described in this document.

  This approach is significantly different from the approach taken in the existing specification, which essentially ignored the issue of authorization of operations on named attribute directories and only described authorization for the OPENATTR operation.

  As there is no known implementation of the earlier approach, it is uncertain whether allowances need to be made for such implementations. In any case, the lack of authorization semantics for named attribute directories means that the treatment of this issue in the existing specifications cannot remain as it is. The precise nature of the advice/recommendations to be provided remains unclear but it will be addressed in Section 12.5 was part of Consensus Item #111.

* The ACL_Choice flag bit ACC4BN_MBCA specifies that when faced with a request to store an ACL that it does not support and is not prepared to enforce the server will sometimes modify it to a different one that it can support, without notice to the client.

  This behavior is considered an allowable behavior variant because existing specifications allowed it. Nevertheless, because of the general uncertainty it creates and its unfortunate effect on interoperability, its use is discouraged as provided for in Section 12.5.

* The ACL_Choice flag bit ACC4BN_SMJUST3 specifies that, when the server is processing a change to the mode, it sets the acl to represent the handling specified by those mode bits, making no effort to preserve other aspects of the ACL

  Although this appears to be an alternative to BN_SMOLD and BN_SMFULL, described below, there are situations in which multiple of these can be true simultaneously

For example, in the case of ACLs that fit within the UNIX ACL model, there are no inheritable ACEs and no ACEs specifying mask bits not controlled by the mode, substituting an ACL consisting of three entries derived from the mode would satisfy BN_SMOLD and BN_SMFULL as well

* The ACL_Choice flag bit ACC4BN_SMOLD specifies that, when the server when processing a change to the mode, it sets the acl to represent the handling specified by those mode bits, while making the efforts to preserve other aspects of the ACL recommended by the existing specifications.

  This recommendation provide for the retention of inherit-only ACEs, but allows effect outside the scope of the mode change in that the elimination of inheritable ACEs affects inheritance as well as authorization for actions involving the current object.

  Although this appears to be an alternative to BN_SMJUST3 and BN_SMFULL, there are situations in which ACL inheritance is not supported so that multiple of these can be true simultaneously.

* The ACL_Choice flag bit ACC4BN_SMFULL specifies that, when the server when processing a change to the mode, it sets the acl to represent the handling of authorization specified by the mode bit, and avoid any other changes to the authorization of actions involving the current object or the authorization for any subordinate objects that result from ACL inheritance.

  Unlike the cases of BN_SMJUST3 and BN_SMOLD, the server’s action is specified in terms of a goal to be to be reached rather than a procedure to be applied. The necessary procedure depends on such matters as support for DENY ACEs and the existence of ACL inheritance and is discussed in Section 10.7.

* The ACL_Choice flag bit ACC4BN_AUTHWHO indicate that the server provides support for special who values relating o the existence of requester authentication/identification.

* The ACL_Choice flag bit ACC4BN_RVINV specifies that the server will, at times, deal with invalid ACLs/ACEs that are normally rejected by accepting them after modifying them to be valid, as provided for by previous specifications.

* The ACL_Choice flag bit ACC4BN_INNO specifies that there is no support for ACL inheritance.

  This bit is incompatible with ACC4BN_IN1BIT, ACC4BN_INHFULL, and ACC4BN_INHAUTO.

* The ACL_Choice flag bit ACC4BN_OTHWHO indicates that the server provides for support for a miscellaneous set of special who values of uncertain semantics.

* The ACL_Choice flag bit ACC4BN_SVTX indicates that the server's authorization of deletion and RENAME is modified based on the directory's setting of MODE4_SVTX, as described in Section 8.3.20.

  In the case in which the objects do not have ACLs, possible because of non-support, the handling is as described in Section 5.3.1 of [I-D.dnoveck-nfsv4-security].

* The ACL_Choice flag bit ACC4BN_SVTXOLD indicates that the server's authorization of deletion and RENAME is modified based on the directory's setting of MODE4_SVTX, as described in Section 6.2.1.3.2 [RFC8881]. Although this approach is outdated and SHOULD NOT be used, previous implementations might have adopted it and continue to use it, bypassing the current recommendation to the contrary.

  In the case in which the objects do not have ACLs, possible because of non-support, the handling is as described in Section 5.3.1 of [I-D.dnoveck-nfsv4-security].

6.7. ACL Choices in Various Stages of Development

The ACL_Choice attribute is designed to be modified over time but the propriety of any particular modification depends on the state of development of the feature. In doing so, we will need to distinguish between extensions and other modifications whose use is more limited.

During development of this specification, until its eventual publication as an RFC, the following stages need to be distinguished:

* The initial specification in this document.

  The main goal is to fully cover the valid semantic range provided by previous specifications where it reasonable to assume that alternate approaches exist. Because of the lax approach taken to the specification of semantics, there could be cases where alternatives are allowed but there is no reason to suppose that the multiple approaches have been adopted.

* As further drafts of this specification are developed there will be the opportunity to provide needed changes.

These will includes the creation of new bits for newly discovered variants and the deletion of bits for variants that were previously allowed but found not to exist.

In addition, cases in which detailed behavior must be specified in a complicated data item could be simplified if a less general approach satisfies the needs of existing implementations.

* Subsequent to Working group last call, the state of the attribute will be permanently assigned to NFSv4.1 and further change will be limited as specified in [RFC8178].

After the completion of the rfc8881bis effort, handling will depend primarily on the minor version, as described below:

* In NFSv4.0, this attribute will not be available. As a result, clients that need the extensions of the UNIX ACL provided by servers that do support will be unable to use them since there is no good way to determine if they are supported. In addition, as described in Section 8.3.19 certain servers whose semantics is allowed by existing specification will not be usable because the semantics associated with certain ACE mask bits might be such that the client cannot map UNIX ACLs to those maintained by the server.

* Similar issues apply to NFSv.1 implementations that were developed before rfc8881bis was published. While it is desirable to update them to provide ACL_Choice attribute, this might not be possible in all cases.

* Many implementations of NFSv4.1 will have support for the ACL_Choice attribute. As a result, the problems that exist for implementations that do not support it will be ameliorated and interoperable operation will be available for clients that do not need any of the extensions to the UNIX ACL model.

* In NFSv4.2, it will be possible make additions to ACL_Choice attribute as extensions to deal with behavioral variants discovered after rfc8881bis was published.

* In a later minor version we will be able to further restrict allowable behavior if the Working Group can reach a consensus. One example would be to make standard the ACE mask bits where the varying-granularity model is workable and expand it cover all mask bits or eliminate mask bits that don't fit. As a result, many existing elements of the ACL_Choice attribute could be made mandatory-to-not-implement and what was left could be limited to the distinct extensions made to the UNIX ACL model to arrive at the NFSv4 ACL model plus a few behavioral variants that the working group decides to accommodate.

7. Structure and Function of Access Control Lists within NFSv4

Subsections within this section and Section 8 describe the structures using to implement a specific sort of security-related semantics. All such semantic currently defined within current NFSv4 minor version fit within this model or a submodel within it. This does not exclude the possibility that addition of similar structures might be defined in later minor versions in order to support other semantic models (see Appendix E.

NFSv4 Access Control Lists consisting of one or more Nfsv4 Access Control Elements. Issues related to the limiting case of an empty list of Nfsv4 Access Control Elements are discussed in Section 5.2. While originally designed to support a more flexible authorization model, these lists have multiple uses within NFSv4, with the use of each element depending on its type, as defined in Section 8.1.

* Access Control Lists may be used to provide a more flexible authorization model as described in Section 8.4 of [I-D.dnoveck-nfsv4-security]. In the NFSv4 ACL model, this involves use of Access Control Entries of the ALLOW and DENY types.

* ACLs may also be used to provide the security-related services described in Section 11. In the NFSv4 ACL model, t his involves use of Access Control Entries of the AUDIT and ALARM types.

[Consensus Needed (Items #61d, #105m, #110c)]: Subsections of this section and of Section 8 define the structure of and semantics of NFSv4 ACLs, whether they are used to represent the UNIX ACLs defined in Section 6.1 or ACLs supporting various extensions thereof supported by the server, which may include any of the extensions defined in this document.

Some matters that relate only to extensions provided to support NFSv4 ACLs are discussed in Section 11 and summarized in Section 8.4 of [I-D.dnoveck-nfsv4-security]. The definition of the NFSv4.1-specific attribute Sacl used only in connections with these functions is provided in Section 5.11.

7.1. Overview of ACL Semantics Choices

[Consensus Needed (Items #105n, #110d), Through end of section]:

For reasons discussed in Section 3, there is an extremely wide range of ACL-related semantics that servers are allowed to support. While it would be desirable to narrow this range, there are sufficient practical problems in doing so that we need to focus on helping clients deal appropriately with this wide range of semantic variability. The following means are available:

* Where supported by the server, the ACL_Choice attribute, described in Section 5.9, can provide extensive information about the ACL semantics provided by the server.

  [Consensus Needed (Item #14c)]: The set of supported ACE types is also needed. Normally, this is provided by the ACL_Support attribute. However, when it is not supported, the default discussed in Section 5.8 is available.

  For a discussion of how the client can best use this information, see Section 7.2.

* Where the ACL_Choice is not supported, information is more limited. This includes the case of use with Minor Version Zero, in which this attribute is not defined.

  For a discussion of how the client can best use the available information, see Section 7.3.

The classes of semantic variability listed below are of potential importance to clients. In some cases, the client’s need for information is related to the client’s own semantic expectations with an important distinction being that between client’s expecting support for the UNIX ACL model and those that require certain features provided by the extensions to that model that servers MAY provide. In the list below, we refer to the former as "UNIX-oriented" clients.

* The set of ACE mask bits supported and how actions are mapped to them.

Although clients might need this information, UNIX-oriented clients can interacts with servers since support for the mask bits needed by these clients is REQUIRED. See Section 8.3.19 for details.

* The presence of authorization-related extensions to the UNIX ACL model such as ACL inheritance and support for DENY ACEs.

  This information is of no interest to UNIX-oriented clients.

* Support for the security-related services described in Section 11.

  This information is of no interest to UNIX-oriented clients.

* The presence of semantics associated with the withdrawn POSIX draft ACLs that we do not treat as part of the UNIX ACL model. These include the way in which a mode is computed from an ACL when an ACL is set (see Section 10.5) and the POSIX-draft-specified handling of the partial satisfaction of ALLOW ACEs in which multiple ACEs can collectively allow an operations even if none of them does so completely.

  Although this information might be of interest to clients when it is available, clients will need to allow for a range of sever behaviors in cases in which ACL_Choice is not supported

* Allowable behavioral variability in the handling of existing ACLs when the mode is set.

  This information is only available to the client when ACL_Choice is supported.

7.2. ACL Semantics Choices

[Consensus Needed (Items #14d, #61e, #105o, #110d), Through end of section]:

There is a wide range of potential ACL semantics that servers can provide. When the ACL_Choice attribute is supported, the client can determine the semantics provided by the server as described below.

* The set of supported ACE mask bits can be determined as described in Section 8.3.17. Also described there is how other elements of the ACE mask support information can be derived from the contents of the ACL_Choice attribute.

* Supported ACE types can be determined using the ACL_Support attribute or the appropriate default value described in Section 5.8.

* The level support for ACL inheritance, including the automatic inheritance features, can be determined as described in Section 8.4.2.

* The way in which the mode corresponding is to be computed based on the objects ACL is that described in Section 10.5. Note that the effect of that procedure for a given file system depends on the ACE mask support information for that file system.

* The way in which existing ACLs are modified when the mode attribute is set is described Section 10.7. Note that the effect of that procedure for a given file system depends on the ACE mask support information for that file system.

* The way is which an ACL being set UNIX-oriented client (which only support three ACE mask bits) can be mapped to an appropriate ACL for the current server is discussed in Section 8.3.19 Note that the effect of that procedure for a given file system depends on the ACE mask support information for that file system..

7.3. Limited Inference Regarding ACL Semantics

In cases in which the ACL_Choice attribute is not supported, including use of the minor version for which it is not defined (i.e. NFSv4.0) there are very limited ways to determine the extensions supported. As a result clients that need to use the extensions to the UNIX ACL are generally unable to determine whether that extension is supported.

It is only possible for clients to use these extensions, if they do so without providing for interoperability, by only using servers that are known, from experience that support the required extensions, in addition to those supporting the ACL_Choice attribute where the set of extensions available can be found as described in Section 7.2.

Although it is possible to test for the presence such extensions, this is generally not helpful because of inherent complexity of making such determinations when crossing into new file systems and because of the freedom given to servers to avoid returning errors for situations in which extensions are used bur not supported.

As will be seen in the following list, paralleling the one in Section 7.2, clients that need to use features beyond those provided by UNIX ACLs are generally limited to use servers supporting ACL_Choice or to using the features in a non-Interoperable way as discussed above.

* The set of supported ACE mask bits in addition to those which are REQUIRED cannot be determined. This is not a problem for UNIX-oriented clients, but those that need to use extensions to that set are basically unable to take advantage of the extensions.

* Supported ACE types can be determined using the ACL_Support attribute or the appropriate default value described in Section 5.8.

* There is no way to determine the level of support for ACL inheritance, including the automatic inheritance features. The is particular troubling since draft-POSIX ACLs require some acl inheritance,

* The way in which the mode corresponding is to be computed based on the objects ACL is that described in Section 10.5. Because that computation depends on information provided by the ACL_Choice attribute, the client winds up having to accept the server’s preferences in this matter.

* The way in which existing ACLs are modified when the mode attribute is set is described Section 10.7. As in the case of computing a mode from an ACL, that computation depends on information provided by the ACL_Choice attribute. As a result, the client winds up having to accept the server’s preferences in this matter.

* The way is which an ACL being set UNIX-oriented client (which only support three ACE mask bits) can be mapped to an appropriate ACL for the current server is discussed in Section 8.3.19. Certain assumptions have to be made about the server’s handling of certain ACE mask bits. However, as long as recommendations in Section 6.3 are adhered to UNIX-oriented clients can function properly.

8. Structure of NFSv4 Access Control Entries

The attributes acl, sacl (v4.1 only) and dacl (v4.1 only) each contain an array of Access Control Entries (ACEs) that are associated with the file system object. The client can set and get these attributes while the server is responsible for using the ACL-related attributes to perform access control. The client can use the OPEN or ACCESS operations to check access without modifying or explicitly

reading data or metadata.

NFSv4 extensions have the option of defining new structures that serve as Access Control Entries for other ACL models.

The NFSv4 ACE structure is defined as follows:

<CODE BEGINS>
```
typedef uint32_t        acetype4;

typedef uint32_t        aceflag4;

struct nfsace4 {
        acetype4        type;
        aceflag4        flag;
        acemask4        access_mask;
        utf8str_mixed   who;
};
```
<CODE ENDS>

8.1. ACE Type

8.1.1. Existing ACE Types

The constants used for the type field (acetype4) and as shifts used in determining ACE type support are as follows:

<CODE BEGINS>
```
const ACE4_ACCESS_ALLOWED_ACE_TYPE      = 0x00000000;
const ACE4_ACCESS_DENIED_ACE_TYPE       = 0x00000001;
const ACE4_SYSTEM_AUDIT_ACE_TYPE        = 0x00000002;
const ACE4_SYSTEM_ALARM_ACE_TYPE        = 0x00000003;
```
<CODE ENDS>

All four are permitted in the Acl attribute. For NFSv4.1 and beyond, only the ALLOWED and DENIED types are used in the Dacl attribute, and only the AUDIT and ALARM types are used in the Sacl attribute.

Value	Abbreviation	Description
ACE4_ACCESS_ALLOWED_ACE_TYPE	ALLOW	Explicitly grants the ability to perform the set of actions specified in acemask4 to the file or directory.

			When all such actions to be done by a given operation are explicitly allowed, the operation is authorized and scanning of the ACL to determine authorization stops.
ACE4_ACCESS_DENIED_ACE_TYPE	DENY		Explicitly denies the ability to perform the set of actions specified in acemask4 to the file or directory. When any of the actions to be done by a given operation are explicitly denied, the operation is unauthorized and scanning of the ACL to determine authorization stops.
ACE4_SYSTEM_AUDIT_ACE_TYPE	AUDIT		Logs (in a system-dependent way) any attempt to perform, for the file or directory, any of the actions specified in acemask4.
ACE4_SYSTEM_ALARM_ACE_TYPE	ALARM		Generates (in a system-dependent way) an alarm upon any attempt to perform, for the file or directory, any of the actions

			specified in
			acemask4.

Table 4

The "Abbreviation" column denotes how the types will be referred to throughout the rest of this document.

8.1.2. ACE Type Support Discovery

[Consensus needed (Item #105p), through end of section]:

Discovery of the ACE types that it is appropriate to use can occur in two ways:

* By use of the OPTIONAL attribute ACL_Support, the set of supported ACE types can be determined.

  This set is limited to ACE types defined in Section 8.1.1 and cannot be easily extended.

* By use of the OPTIONAL attribute ACL_Choice, the set of supported ACE types can be determined using the data item ACC4IN_TSUPP, when that item is present.

  This set can include ACE types defined by protocol extensions as described in Section 8.1.3 as well as those defined in Section 8.1.1.

When neither of these attribute values are available, the client has no way of determining the ACE types supported and when attempting the use of ACE types other than ALLOW needs to be prepared for a failure due to non-support. Similarly, when the Aclfeature attribute is not supported, clients attempting to use ACE types other than those defined in Section 8.1.1, need to be prepared for failure to be returned due to non-support.

8.1.3. ACE Type Extension

[Consensus needed (Item #105q), through end of section]:

Standards-track documents which define NFSv4 protocol extensions, as provided for in [RFC8178], can extend the set of ACE types. The definition of a new extension type needs to provide the following information:

* An ACE type name, generally of the form ACE4_???_????_ACE_TYPE which is to be used to define ACEs of the specified type, when appearing in the type fields of the ACE.

* A numeric value between four and nine that has not been previously used as an ACE type value.

  While it is theoretically possible to delete a previously defined ACE type as part of a new minor version, the practical difficulties that result from these being stored within existing file systems require that such numeric values not be reused.

* An abbreviation to be used when referring to that ACE type.

* A description of the effect of the ACEs of the specified type within ACL. This needs to include, for ACE types that can appear within existing ACL-based attributes, how the presence of the ACE affects existing scans of ACL-based attributes such as an authorization scan or a notification scan in response to action successfully or unsuccessfully attempted.

* Description of the set of attributes in which the ACE type can appear which can include Acl, Dacl, Sacl, and new attributes added in the same or previous extensions.

8.2. ACE Inheritance

[Consensus needed (Item #127c), Through start of included subsection]: It is often desirable to control access to files within a directory hierarchy so that particular subtrees within the hierarchy share ACEs that allow, deny, or provide specific monitoring for sets of files with a common ancestor. Despite the need for this function, server file systems would find it difficult to find the controlling ACEs if they are associated only with the encompassing directory.

In order to avoid such a need, Windows ACLs (and the NFSv4 ACLs based on them), provide ways for such ACEs to be propagated to lower levels of the naming hierarchy in twoways:

* By establishing inheritable ACEs in the encompassing directories within the server filesystem responsible for propagation of these as subordinate objects are created.

  This deals with situations in which the object is created after the ACEs to be applied are established.

This form of propagation is effected by the server filesystem. It is described in more detail in Section 8.2.1.

* By providing for later propagation of ACEs when the ACEs to be applied are created, modified or deleted.

  This form of propagation is effected by the client in response to ACE changes and changes in the naming hierarchy resulting from RENAMEs.

  It is described in more detail in Section 8.2.2.

8.2.1. Server-effected ACE inheritance

[Consensus needed (Item #127d), through end of section]: Whether a given ACE is subject to this form of inheritance is controlled by the ACE flag bits ACE4_FILE_INHERIT_ACE and ACE4_DIRECTORY_INHERIT_ACE. Whenever either of these bits is specified for an ACE associated with a directory, the ACE will considered for inheritance when subordinate objects are created within that directory.

Such ACEs are considered in making authorization and monitoring decisions if ACE4_INHERIT_ONLY_ACE is not set.

8.2.2. Client-managed ACE inheritance

[Consensus needed (Item #127e), t\Through end of section]: In order to ensure that objects lower in the name hierarchy, always reflect the inheritable ACEs in the encompassing objects they are part of:

* The user can periodically update the ACEs for subordinate objects explicitly.

* The client performs the necessary updates when the set of inheritable ACEs change or when the structure of the naming hierarchy changes as a result of RENAME operations.

  Because this method arose in the Windows context as an alternative to requiring the user to do the updates, it was described there as "automatic" inheritance. This is despite the fact that, when considered together with server-effected inheritance, neither appears more "automatic" than the other.

8.2.3. Details of "Automatic" Inheritance

[Consensus needed (Item #127f), Through end of section]: In order to provide support for client-managed update of inheritable ACEs. the sacl (Section 5.11) and dacl (Section 5.10) attributes include an additional flag field preceding the ACE array.

```
<CODE BEGINS>
struct nfsacl41 {
        aclflag4        na41_flag;
        nfsace4         na41_aces<>;
};
<CODE ENDS>
```

The flag field applies to the entire sacl or dacl; the following flag values are defined:

```
<CODE BEGINS>
const ACL4_AUTO_INHERIT         = 0x00000001;
const ACL4_PROTECTED            = 0x00000002;
const ACL4_DEFAULTED            = 0x00000004;
<CODE ENDS>
```

All bits not defined above are to be cleared. The ACE4_INHERITED_ACE flag can be set in the ACEs of the sacl or dacl (whereas it always needs to be cleared in the acl). These bits provide support for the two inheritance-related features listed below. See Section 8.4.2 for information about how support for each feature can be ascertained.

* ACL4_AUTO_INHERIT, ACL4_PROTECTED, and ACL4_DEFAULTED are used to support automatic inheritance, allowing the changes to the ACLs for objects higher in directory hierarchy to be selectively propagated to objects which have inherited ACEs from ACLs for directories above them in the namespace.

  These flags are stored together with the ACEs making up the ACL and are returned when the sacl or dacl attribute is interrogated

These flags have the following meanings:

* The flag ACL4_AUTO_INHERIT indicates that the set of ACEs is subject to later modification to reflect the propagation of ACL changes to lower levels of the directory hierarchy.

* The flag ACL4_PROTECTED indicates that the set of ACES is not to subject to later modification to reflect the propagation of ACL changes to lower levels of the directory hierarchy.

* The flag ACL4_DEFAULTED indicates that the set of ACEs is derived from ACL inheritance and does not include ACEs specified by the creator of the object. As a result, the propagation of ACL changes to lower levels of the directory hierarchy does not need to concern itself with the possible effects of such creator-selected ACEs.

The elements used allow a server to support automatic inheritance are explained in more detail below.

Inheritable ACEs are normally inherited by child objects only at the time that the child objects are created; later modifications to inheritable ACEs do not result in modifications to inherited ACEs on descendants.

However, the dacl and sacl attributes provide the ability to have an ACL inheritance mechanism that allows a client application to propagate changes to inheritable ACEs throughout an entire directory hierarchy.

A server that supports this feature performs inheritance at object creation time in the normal way, and sets the ACE4_INHERITED_ACE flag on any inherited ACEs as they are added to the new object.

A client application such as an ACL editor can then propagate changes to inheritable ACEs on a directory by recursively traversing that directory’s descendants and modifying each NFSv4 ACLs encountered to remove any ACEs with the ACE4_INHERITED_ACE flag and to replace them by the new inheritable ACEs (also with the ACE4_INHERITED_ACE flag set). It uses the existing ACE inheritance flags in the obvious way to decide which ACEs to propagate. (Note that it may encounter further inheritable ACEs when descending the directory hierarchy and that those will also need to be taken into account when propagating inheritable ACEs to further descendants.)

The reach of this propagation may be limited in two ways:

* Automatic inheritance is not performed from any directory ACL that has the ACL4_AUTO_INHERIT flag cleared.

* Automatic inheritance stops wherever an ACL with the ACL4_PROTECTED flag is set, preventing modification of that ACL and also (if the ACL is set on a directory) of the ACL on any of the object’s descendants.

This propagation is performed independently for the sacl and the dacl attributes; thus, the ACL4_AUTO_INHERIT and ACL4_PROTECTED flags may be independently set for the sacl and the dacl, and propagation of one type of acl may continue down a hierarchy even where propagation of the other acl has stopped.

New objects are to be created with a dacl and a sacl that both have the ACL4_PROTECTED flag cleared and the ACL4_AUTO_INHERIT flag set to the same value as that on, respectively, the sacl or dacl of the parent object.

Both the dacl and sacl attributes are OPTIONAL. However, if a server supports one of these, it MUST support the other if it supports any of the ACE types are assigned to that attribute.

A server that supports both the acl attribute and one or both of the new dacl or sacl attributes MUST do so in such a way as to keep all three attributes consistent with each other. Thus, the ACEs reported in the acl attribute will be the union of the ACEs reported in the dacl and sacl attributes, except that the ACE4_INHERITED_ACE flag will be cleared from the ACEs in the acl. And of course a client that queries only the acl will be unable to determine the values of the sacl or dacl flag fields.

When a client performs a SETATTR for the acl attribute, the server sets the ACL4_PROTECTED flag to true on both the sacl and the dacl. By using the acl attribute, as opposed to the dacl or sacl attributes, the client signals that it might not understand automatic inheritance, and thus cannot be trusted to set an ACL for which automatic inheritance would make sense.

When a client application queries an NFSv4 ACL, modifies it, and sets it again, it needs to leave any ACEs marked with ACE4_INHERITED_ACE unchanged, in their original order, at the end of the NFSv4 ACL. If the application is unable to do this, it needs to set the ACL4_PROTECTED flag. This behavior is not enforced by servers, but violations of this rule may lead to unexpected results when applications perform automatic inheritance.

When the low-order bits of mode attribute are subject to modification, the server will, if possible, set the mode in such a way that leaves inherited ACEs unchanged, in their original order, at the end of the ACL. If it is unable to do so, it sets the ACL4_PROTECTED flag on the file’s dacl.

Finally, in the case where the request that creates a new file or directory does not also set permissions for that file or directory, and there are also no ACEs to inherit from the parent’s directory,

then the server’s choice of ACL for the new object is implementation-dependent. In this case, the server is to set the ACL4_DEFAULTED flag on the ACL it chooses for the new object. An application performing automatic inheritance takes the ACL4_DEFAULTED flag as a sign that the ACL is to be completely replaced by one generated using the automatic inheritance rules.

8.3. ACE Access Mask

[Consensus needed (Item #126a), Through start of included subsections]: Section 8.3.1 describes the individual bits within the ACE access mask, each of which designates a set of actions that can be allowed, denied, otherwise monitored based on the ACE entries that include various of these mask bits. Sections 8.3.4 through 8.3.10 provide more detail on each of these bit masks

8.3.1. ACE Access Mask

[Consensus needed (Item #126b), through end of section]: The following bitmask constants can be used within the access mask field of the ACE. For references within the c++-oriented pseudocode in Sections 8.3.14 through 8.3.16, there will be a corresponding enum value within AceMask::Values.

[Consensus Needed (Items #121d, #122c), Through end of sourcecode section]:

```
<CODE BEGINS>
const ACE4_READ_DATA            = 0x00000001;
const ACE4_LIST_DIRECTORY       = 0x00000001;
const ACE4_WRITE_DATA           = 0x00000002;
const ACE4_ADD_FILE             = 0x00000002;
const ACE4_APPEND_DATA          = 0x00000004;
const ACE4_ADD_SUBDIRECTORY     = 0x00000004;
const ACE4_READ_NAMED_ATTRS     = 0x00000008;
const ACE4_WRITE_NAMED_ATTRS    = 0x00000010;
const ACE4_EXECUTE              = 0x00000020;
const ACE4_DELETE_CHILD         = 0x00000040;
const ACE4_READ_ATTRIBUTES      = 0x00000080;
const ACE4_WRITE_ATTRIBUTES     = 0x00000100;
const ACE4_WRITE_RETENTION      = 0x00000200;
const ACE4_WRITE_RETENTION_HOLD = 0x00000400;

const ACE4_DELETE               = 0x00010000;
const ACE4_READ_ACL             = 0x00020000;
const ACE4_WRITE_ACL            = 0x00040000;
const ACE4_WRITE_OWNER          = 0x00080000;
const ACE4_SYNCHRONIZE          = 0x00100000;
const ACE4_READ_EXT_ATTRS       = 0x00200000;
const ACE4_WRITE_EXT_ATTRS      = 0x00400000;
const ACE4_SACL_ACCESS          = 0x00800000;
<CODE ENDS>
```

Note that some masks have coincident values, or are treated differently when used with different types of objects. For example, ACE4_READ_DATA and ACE4_LIST_DIRECTORY designate the same mask bit which is treated differently depending on whether the object is a directory or other type of object. Note that,

* The mask value names ACE4_ADD_FILE, ACE4_ADD_SUBDIRECTORY, and ACE4_DELETE_CHILD are intended to be used with directory objects that are not named attribute directories and are not supported when used with objects of other types.

* The mask value name ACE4_APPEND_DATA is intended to be used with non-directory objects.

* The mask values used for ACE4_READ_DATA and ACE4_LIST_DIRECTORY designate the same mask which treated differently depending on whether the object is a directory which is not a named attribute directory, or other type of object.

  The mask values for ACE4_WRITE_DATA and ACE4_ADD_FILE behave similarly. as do the mask values for ACE4_APPEND_DATA and ACE4_ADD_SUBDIRECTORY. In each case, the same mask bit has two

different names and controls two different sets of actions, depending on whether the underlying object is a directory which is not a named attribute directory or another type of object.

* The mask bit designated by ACE4_EXECUTE controls two different sets of action depending on whether the underlying object is a directory which is not a named attribute directory or another type of object.

[Consensus Needed (Items #102a, #103a), through end of list]: These mask bit are explained in more detail in the sections mentioned below based on their relationship to the three POSIX-derived permission bits: Read, Write, and Execute. As described in Section 8.3.4, it is no longer assumed that different sets of supported mask bits can be adequately explained as a different server choice as to mask bit "granularity". Changes include material in multiple subsections of Section 8.3.

* Mask bits whose set of authorized actions corresponds to a single POSIX-derived permission bit are explained in Section 8.3.5.

  These mask bits are always to be supported although the set of authorized actions is expected to be smaller when other mask bits covering a smaller set of actions are supported.

* Mask bits whose set of authorized actions is a subset of those normally controlled by a single POSIX-derived permission bit are explained in Section 8.3.6. These only include mask bits controlled by the write privilege bit.

  These mask bits are not always supported, but depend on ACL extensions supported by the server. For detailed guidance regarding how the client can determine which mask bits are supported, see Sections 7.2 and 7.3.

* Mask bits whose set of authorized actions is a subset of those normally controlled by the two POSIX-derived permission bits named X and R are explained in Section 8.3.8. These two bit often control actions necessary to access the file data but do not distinguish, as the X and R bits do for non-directory objects reading the file for execution and for other purposes.

* Mask bits whose set of authorized actions is a subset of those normally allowed to the owner of a file are explained in Section 8.3.9.

These mask bits are not always supported, but depend on ACL extensions supported by the server. For detailed guidance regarding how the client can determine which mask bits are supported, see Sections 7.2 and 7.3.

* Other mask bits are explained in Section 8.3.10. These include mask bits which do not fit in the varying-granularity approach since they cannot be considered finer-grained variants of either a single POSIX privilege bit, two closely related privilege bits, or of ownership of the file. Also included are bits where the relationship to POSIX privileges is uncertain and might depend on the behavior of particular implementations.

  This section includes some mask bits for which we have found existing implementations and some for which no implementations have yet been found. These mask bits are not always supported, but clients need to be prepared for support actually present depending on the set of ACL extensions supported which can be ascertained by the client when the server supports the ACL_Choice attribute

[Consensus Needed (Item #5a) The descriptions in the section below are relevant to both authorization and for recognizing operations whose success or failure are to be recorded when ACLs are used to provide the non-authorization functions described in Section 11. With regard to use of ACCESS whose returned bits are to be affected, it is not necessarily the case that the occurrence of ACCESS in these lists which specify those bits implies that such operation are to be treated as recordable events. This because no ACCESS only provides information and does not attempt access.

[Consensus Needed (Item #4c)]: While it s recommended that The sets of actions to be authorized or otherwise noted in connection with these mask bits be those cited in the sections below, it is possible that existing implementations might behave differently, based on their reliance on earlier specifications and a common understanding within the working group that it was the job of the specification to conform to the implementation, rather than the other way around. See Section 8.3.17 for information about how the client is to be made aware of such discordant implementations.

8.3.2. Changes in Descriptions of Mask Bits

[Author Aside, Through end of section]: The material in this section identifies changes it has been necessary to make in the description of the ACE mask bits. It is likely that it will be removed before the successor document is published as an RFC

The following items should be noted as cases in which a change related to the description of ACE mask bits. In some cases, there will be corresponding annotations near the actual text change, but this is not always the case. Nevertheless, there will need to be consensus regarding the following changes:

* [Author Aside (Item #3a)]: Because the following sections have been moved to be part of a general description of ACEs, not limited to authorization, the descriptions no longer refer to permissions but rather to actions. This could be considered a purely editorial change, but, to allow for possible disagreement on the matter, it will be considered, here and in Appendix C, as consensus item #3.

* [Author Aside (Item #4d)]: In a large number of places, SHOULD was used inappropriately, since there appear to be no valid reasons to allow a server to ignore what might well be a requirement. Such changes are not always noted individually below. However, they will be considered, here and in Appendix C, as part of consensus item #4.

* [Author Aside (Item #5)}: In a significant number of cases the ACCESS operation had not been listed as an operation affected by the mask bit where logic suggests it needs to be. These individual additions are not noted individually below, although there is, in each affected section, an annotation indicating that section requires consensus on this point. In all cases, they will be considered, here, in the affected sections and in Appendix C, as part of consensus item #5.

  When ACCESS is included as an affected operation, the description identifies the returned bits that are to affected.

  When ACCESS is listed as affected, this is only with regard to authorization. Non-authorization uses are discussed elsewhere, as part of this consensus item.

* [Author Aside, Including entire bulleted item]: In a number of cases, there are additional changes which go beyond editorial or arguably do so. These will be marked as their own consensus items usually with an accompanying author aside but without necessarily citing the previous treatment. These include the following:

  [Author Aside (Item #7a)]: Revisions were necessary to clarify the relationship between READ_DATA and EXECUTE.

[Author Aside (Item #8a)]: Revisions were necessary to clarify the relationship between WRITE_DATA and APPEND_DATA. These are part of consensus item #8.

[Author Aside (Item #9a)]: Clarification of the handling of RENAME by ADD_FILE, ADD_SUBDIRECTORY.

* Revisions in handling of the masks WRITE_RETENTION and WRITE_RETENTION_HOLD. These are parts of consensus items #10.

8.3.3. Role of Sticky Bit in ACL-based Authorization

[Author Aside (Item #62b)]: Because of the need to address sticky-bit issue as part of the ACE mask descriptions, it is appropriate to introduce the subject here.

[Author Aside (Item #62b)]: Despite the fact that NFSv4 ACLs and mode bits are separate means of authorization, it has been necessary, even if only for the purpose of providing compatibility with earlier implementations, to introduce the issue here, since reference to this mode bit are necessary to resolve issues regard directory entry deletion, as is done in Section 8.3.20.

[Consensus Item, Including List (Item #62b): The full description of the role of the sticky-bit appears in Section 5.3.2 of [I-D.dnoveck-nfsv4-security]. In evaluating and understanding the relationship between the handling of this bit when NFSv4 ACLs are used and when they are not, the following points need to be kept in mind:

* This is troublesome in that it combines data normally assigned to two different authorization models and breaks the overall architectural arrangement in which the mask bits represent the mode bits but provide a finer granularity of control.

* It might have been possible to conform to the existing architectural model if a new mask bit were created to represent the directory sticky bit. It is probably too late to do so now, even though it would be allowed, from the protocol point of view, as an NFSv4.2 extension.

* The new treatment in Section 8.3.20 is more restrictive than the previous one appearing in Appendix B.1. This raises potential compatibility issues since the new treatment, while designed to address the same issues was designed to match existing Unix handling of this bit.

* This handling initially addresses REMOVE and does not address directory sticky bit semantics with regard to RENAME. Whether it will do so is still uncertain.

* The handling of this mode bit was not documented in previous specifications. However, there is a preliminary attempt to do so in Section 5.3.1 of [I-D.dnoveck-nfsv4-security]. The reason for doing so, is that, given the Unix orientation of the mode attribute, it is likely that servers currently implement this, even though there is no NFSv4 documentation of this semantics

  This treatment needs to be checked for compatibility issues and also to establish a model that we might adapt to the case of NFSv4 ACLs.

* In the long term, it would make more sense to allow the client rather than the server to have the primary role in determining the semantics for things like this. That does not seem possible right now but it is worth considering.

8.3.4. Handling of OPTIONAL Mask Bits

In previous specifications, it was assumed that, given the finer granularity of the ACE mask word compared to the three POSIX permission bits the difference between these two approaches could be usefully summarized as the result of different levels of granularity, with each mask bit considered either equivalent to a POSIX permission bit or a finer-grained subdivision of one such bit. See Section 3.5 for explanations of the issues that arose from this approach when it was assumed that this was a full description and that it was the responsibility of the server.

In this document, we divide discussion of the ACE mask word according to the granularity refinement relationships which actually exist, without assuming that are as neat as assumed previously.

8.3.5. Uses of Core Mask Bits

[Consensus Needed (Items #4e, #5b, #7b, #8b, #106a, #107a, #108a, #109a), Throughout section]

ACE4_READ_DATA (for non-directory objects)

Operation(s) affected:
READ

[Consensus Needed (Item #101a)]: READLINK

OPEN (for read or read-write)

ACCESS (ACCESS4_READ)

Discussion:
The action of reading the data of the file, or, in some cases, providing necessary preparation to do so.

[Previous Treatment (Items #4e, #7b)]: Servers SHOULD allow a user the ability to read the data of the file when only the ACE4_EXECUTE access mask bit is allowed.

[Author Aside (Item #7b)]: The treatment needs to be clarified to make it appropriate to all ACE types.

[Consensus Needed (Items #4e, #7b)]: When used to handle READ or OPEN operations, the handling MUST be identical whether this bit, ACE4_EXECUTE, or both are present, as the server has no way of determining whether a file is being read for execution are not. The only occasion for different handling is in construction of a corresponding mode or in responding to the ACCESS operation.

ACE4_LIST_DIRECTORY (for ordinary directories)

Operation(s) affected:
READDIR

ACCESS (ACCESS_READ)

Discussion:
The action of enumerating the contents of a directory, as opposed to searching for a particular name.

ACE4_ADD_FILE(for ordinary directories)

Operation(s) affected:
CREATE

- Will require ACE4_ADD_SUBDIRECTORY to add a directory , when this is supported.

LINK

OPEN (which creates file in the directory)

ACCESS (ACCESS4_EXTEND)

REMOVE (may require ACE4_DELETE_CHIILD, when this is supported

RENAME (on the target directory)

Discussion:
Operations which modify a directory

Many of these operations may controlled at a finer granularity, when the appropriate mask bits are supported.

ACE4_WRITE_DATA (for all non-directory object types)

Operation(s) affected (only when ACE4_WRITE_NAMED_ATTRS is not supported):
OPENATTR (when createdir is true)

Discussion:
[Consensus Needed (Item #111b)]: Through rest of this entry

This action is normally controlled by ACE4_WRITE_NAMED_ATTRS, but needs to be controlled by ACE4_WRITE_DATA when the former is not supported (e.g. when UNIX ACLs without additional ACE mask support are provided by the server).

ACE4_EXECUTE (for non-directory objects)

Operation(s) affected:
READ

OPEN (for read or read-write)

ACCESS (ACCESS4_EXECUTE)

Discussion:
The action of reading a file in order to execute it.

Servers MUST allow a user the ability to read the data of the file when only the ACE4_EXECUTE access mask bit is allowed. This is because there is no way to execute a file without reading the contents. Though a server may treat ACE4_EXECUTE and ACE4_READ_DATA bits identically when deciding to permit a READ or OPEN operation, it MUST still allow the two bits to be set independently in NFSv4 ACLs, and distinguish between them when replying to ACCESS operations. In particular, servers MUST NOT silently turn on one of the two bits when the other is set, as that would make it impossible for the client to correctly enforce the distinction between read and execute permissions.

As an example, following a SETATTR of the following NFSv4 ACL:

nfsuser:ACE4_EXECUTE:ALLOW

A subsequent GETATTR of acl attribute for that file will return:

nfsuser:ACE4_EXECUTE:ALLOW

and MUST NOT return:

nfsuser:ACE4_EXECUTE/ACE4_READ_DATA:ALLOW

ACE4_EXECUTE (for directories which are not named attribute directories)

Operation(s) affected:
LOOKUP

ACCESS(ACCESS4_LOOKUP)

REMOVE

RENAME

Discussion:
The action of traversing directory by searching for a particular named item.

8.3.6. Finer-grained Mask Bits Derived from Write

[Consensus Needed (Item #103b), Through the end of the following list]: The mask bits presented in this section all control some subset of the actions controlled by the write permission bit when POSIX authorization is in effect. These mask bits, when fully supported, provide for finer-grained control of authorization decisions. The corresponding ACE mask bit, ACE4_WRITE_DATA, still controls actions for which no corresponding mask bit defined in this section provides for finer-grained control or where the defined bit is not supported by a particular server implementation.

Although object deletion is controlled by ACE4_DELETE for all types of objects, the situation is different for directories and for non-directory objects:

* For non-directory objects, once file deletions are excluded, all actions can be divided into those controlled by ACE4_WRITE_DATA and those controlled by ACE4_APPEND_DATA.

Because of differences among the description of the use of these mask bits in previous specifications, the implementation of the corresponding bits in Windows, and the semantics of OPEN/WRITE in NFSv4, the way in which these bits are to be used is complicated matter best discusssed elsewhere.

These matters are dealt with in Section 8.3.7. Because of limitations on protocol changes within bis documents, the discussion there will probably need supplementation in later minor versions as discussed in Appendix F.

* For directories, many actions are subject to finer-grained control when the mask bits defined in this section are implemented. These include ACE4_ADD_SUBDIRECTORY and ACE4_DELETE_CHILD.

  [Author Aside (Item #9b)]: The handling of RENAME in distinguishing cross-directory and within-directory RENAME options has to be considered tentative. However, even though this is different from previous treatments of the issue, it needs to be carefully considered by the working group. This is primarily because there seems to be no clear motivation for the previous treatment and because it seems unlikely that restrictions on adding or deleting objects would necessitate corresponding restrictions on renaming them, in case in which the directory was not read-only.

  [Consensus Needed (Item #9b)]: Even when all the above bits are fully supported, the action of renaming a file or directory is controlled by ACE4_ADD_FILE for the enclosing directory since this mask bit is the corelate of the write privilege bit for a directory.

[Consensus Needed (Items #4f, #5c, #7b, #8b, #101a, #106a, #107a, #108a, #109a, #121e), Throughout rest of section]

ACE4_ADD_FILE (For directories)

Operation(s) affected:

CREATE

LINK

OPEN (when creating a new file)

RENAME (in the cross-directory case)

Discussion:
The action of adding a new file in a directory. The CREATE operation is affected when nfs_ftype4 is NF4LNK, NF4BLK, NF4CHR, NF4SOCK, or NF4FIFO. (NF4DIR is not included because it is covered by ACE4_ADD_SUBDIRECTORY.) OPEN is affected when used to create a regular file. LINK is always affected and RENAME is affected when a file/directory is moved between directories, with ACE4_ADD_SUBDIRECTORY covering the case when a directory is renamed between directories.

ACE4_APPEND_DATA (For non-directory objects)

Operation(s) affected:
WRITE

ACCESS

OPEN

SETATTR of size

Discussion:
[Author Aside]: Also needs to be revised to deal with issues related to the interaction of WRITE_DATA and APPEND_DATA.

The action of modifying a file’s data, but only starting at EOF. This allows for the specification of append-only files, by allowing ACE4_APPEND_DATA and denying ACE4_WRITE_DATA to the same user or group.

[Consensus Needed (Item #8c)]: As there is no way for the server to decide, in processing an OPEN or ACCESS request, whether subsequent WRITEs will extend the file or not, the server will treat masks containing only WRITE_DATA, only APPEND_DATA or both, identically.

[Consensus Needed (Item #8c)]: If the server is processing a WRITE request and the area to be written extends beyond the existing EOF of the file then the state of APPEND_DATA mask bit is consulted to determine whether the operation is permitted or whether alarm or audit activities are to be performed. If a file has an NFSv4 ACL allowing only APPEND_DATA (and not WRITE_DATA) and a WRITE request is made at an offset below EOF, the server MUST return NFS4ERR_ACCESS.

[Consensus Needed (Item #8c)]: If the server is processing a WRITE request and the area to be written does not extend beyond the existing EOF of the file then the state of APPEND_DATA mask

bit does not need to be consulted to determine whether the operation is permitted or whether alarm or audit activities are to be performed. In this case, only the WRITE_DATA mask bit needs to be checked to determine whether the WRITE is authorized.

ACE4_ADD_SUBDIRECTORY (For directories)

Operation(s) affected:
CREATE

RENAME (in the cross-directory case)

Discussion:
[Author Aside]: The RENAME cases need to be limited to the renaming of directories, rather than saying, "The RENAME operation is always affected."

[Consensus Needed (Item #9b)]: The action of creating a subdirectory in a directory. The CREATE operation is affected when nfs_ftype4 is NF4DIR. The RENAME operation is always affected when directories are renamed and the target directory NFSv4 ACL contains the mask bit ACE4_ADD_SUBDIRECTORY.

When this bit is not supported, the actions normally controlled by it are controlled by ACE4_ADD_FILE.

When the ACL_Choice attribute is supported, support for the attribute is reported using the bit ACC4BN_SEPAFD.

ACE4_DELETE_CHILD (For Directories)

Operation(s) affected:
REMOVE

RENAME (in the cross-directory case)

Discussion:
The action of deleting a file or directory within a directory.

See Section 8.3.20 for information on now ACE4_DELETE and ACE4_DELETE_CHILD are to interact.

When this bit is not supported, the actions normally controlled by it are controlled by ACE4_ADD_FILE.

When the ACL_Choice attribute is supported, support for the attribute is reported using the bit ACC4BN_SETDE.

ACE$_WRITE_EXT_ATTRS

Operation(s) affected (outside NFSv4)
When used by filesystems that support Windows Extended Attribute, any operation which modifies such an attribute.

Operation(s) affected (within NFSv4.2)
SETXATTR

ACE4_WRITE_NAMED_ATTRS

Operation(s) affected, for all object types:
OPENATTR (when createdir is true)

Operation(s) affected, for the associated named attribute directory:
CREATE

LINK

OPEN (which creates file in the directory)

ACCESS (ACCESS4_WRIzTE, ACCESS4_EXTEND)

REMOVE

RENAME

Discussion:
The action of writing to named attributes of a file or of creating a named attribute directory. OPENATTR is affected when it is used to create a named attribute directory. This is when createdir is TRUE.

[Author Aside (Item #111c)]: Despite the original intention for this bit, it appears that the scheme originally specified incorrectly allowed all sorts of changes to named attributes as long as the named attribute directory already existed.

The ability to check whether or not a named attribute directory exists depends on the ability to look it up; therefore, users also need the ACE4_READ_NAMED_ATTRS permission in order to create a named attribute directory.

[Author Aside (Item #111c)]: Actually, they need that permission to create a named attribute.

When this bit is not supported, the actions normally controlled by it are controlled by ACE4_ADD_FILE.

When the ACL_Choice attribute is supported, support for this mask bit is reported using the bit ACC4BN_SEPWNA.

8.3.7. Distinguishing WRITES Covered by the Append Mask Bit

The handling of appending WRITEs has become a troublesome issue because of a number of areas related to its design were dealt with separately, with no coordination among them.

(1): The original descriptions of these mask bits (in [RFC3530] and pretty much unchanged in [RFC8881]) contrasts "the ability to append to a file with "the ability to modify existing contents, suggesting, particularly given that ACLs are basically a security-oriented feature that the ability to modify existing data has a reasonable basis to be addressed by a separate mask bit, with the addition of additional data to a file inherently different from a security point of view.

This is despite the fact that the motivation for this separate bit is now known to arise from a different basis reflecting Windows Append semantics. which outside the scope of POSIX WRITE support. While conversion of NFSv4 support to incorporate this aspect of Windows Semantics might have been considered when the original RFC was being drafted it is not possible now because NFSv4 does not support the needed semantics to support append-only OPENs and the earliest such changes could be added is in NFSv4.2.

(2): The actual Windows implementation on which this mask bit was based took a very different approach, which we would normally adopt now while making allowances for those relying on existing specifications that followed the mistaken approach described above. We cannot do that now for reasons discussed below.

NFSv4 OPEN was never enhanced with an Append bit, either in NFSv4.0 or NFSv4.1. In addition, WRITE retained its basic model, derived from POSIX, allowing the write point to be specified for each WRITE, with no attention to other models which might have been adopted to provide support for appending to files from multiple clients.

To deal with this confused situation, there are a number of possible ways to accommodate this situation. One, referred to below as (2a) would treat APPEND like SYNCHRONIZE (i.e., not enforceable by NFSv4 but retained and managed by NFSv4 for the

use of other remote access protocols or local access). Another would accommodate NFSv3/v4 access until NFSv4 support for append OPENs is available in a subsequent minor version. This approach, referred to below as (2b), would allow of use the APPEND bit for protocols without specific append support as long as the protocol used had no support for append-only OPENs. The check would be like that discussed in (1) with the proviso that gaps between the EOF and the write point would be disabling, since the security of existing data is guaranteed even though this is troubling for effective writing of log files.

(3): Because POSIX has support for Append-oriented opens, even though it appears that it cannot be a basis for a security-related feature, there are difficulties to address in using that flag in connection with testing the ACE mask bit for APPEND.

This is because it only makes it easier to append-only files, does not enforce their use, with the case of multiple OPENs on different clients not really addressed.

(4): Because NFSv4 is an extensible protocol, it is possible to extend it to provide Windows-like Append semantics as discussed in Appendix F.3

Since the client needs a way to determine the security implications, the choice descried above together with some details needs to be made available to the client. The use of a data structure within the ACL_Choice attribute is described in Section 12.3. Although the choices are not organized as described above, Section 12.3 will explain how these choices are represented using the ACL_Choice attribute.

8.3.8. Finer-grained Mask Bits Derived from Read/Execute

[Consensus Needed (Items #103c, #121f), Through end of section]:

ACE4_READ_EXT_ATTRS (for all object types)

Operation(s) affected (for v4.2 xattr extension):
LISTXATTR

GETXATTR

Discussion:

When referenced by NFS4.1 server, which have no support for extend attributes these bits are stored and returned but do not control authorization. Thet are available for use by locally access and other remote access protocols that do have support for extended attributes

Within NFSv4.2, applies to use of the xattr extension.

May be used by future extensions that provide a Windows-like extended attributes extension.

ACE4_READ_NAMED_ATTRS (for all object types)

Operation(s) affected:
OPENATTR

OPEN when the current fh is a named attribute directory(for read or read-write)

ACCESS when the current fh is a named attribute directory (ACCESS4_READ, ACCESS4_LOOKUP)

READDIR when the current fh is a named attribute directory

LOOKUP when the current fh is a named attribute directory

Discussion:
The action of reading the named attribute directory of a file or of looking up a named attribute directory.

[Author Aside (Item #111d)]: It appears that this bit is identical to the or of the read and execute privilege bits for the named attribute directory. Unfortunately, there is no way to set the Mode attribute (or any attribute) of the named attribute directory.

[Consensus needed (Item #111d): When this mask bit is not supported, the actions normally controlled by it are controlled by the or of the two mask bits ACE4_EXECUTE and ACE4_LIST_DIRECTORY.

[Previous treatment (Item 111d)}: OPENATTR is affected when it is not used to create a named attribute directory. This is when 1) createdir is TRUE, but a named attribute directory already exists, or 2) createdir is FALSE.

[Author Aside (Item #111d)]: The implication in the above paragraph, the OPENATTR is not affected when it is used to create a named directory is troublesome, since, regardless of the clients intentions, lookup access in the named directory should probably be necessary to look up things within it.

8.3.9. Finer-grained Mask Bits Derived from Ownership

The mask bits discussed in this section all authorize actions, that, in the absence of support for that bit are resolved based on the ownership of the file object.

When the ACL_Choice attribute is supported, the support for each of these bits is normally indicated using the data item ACC4IN_OWNMB. However, because the existing specifications do no require this behavior, it is possible that server may not use that model. In such cases, and when support is different for the directory and non-directory cases, support is reported using the data item ACC4IN_ODDMB.

ACE4_WRITE_ATTRIBUTES (for all object types)

Operation(s) affected:
SETATTR of time_access_set, time_backup, time_create, time_modify_set, mimetype, hidden, system.

Discussion:
The action of changing the times associated with a file or directory to an arbitrary value. Also controls changing the mimetype, hidden, and system attributes.

Implementation Information:
A user having ACE4_WRITE_DATA or ACE4_WRITE_ATTRIBUTES will be allowed to set the times associated with a file to the current server time.

ACE4_WRITE_ACL (for all object types)

Operation(s) affected:
SETATTR of acl and mode

Discussion:
The action of modifying the acl or mode attributes.

ACE4_WRITE_OWNER (for all object types)

Operation(s) affected:
SETATTR of owner and owner_group

Discussion:
The action of modifying the owner or owner_group attributes. On UNIX systems, this done by executing chown() and chgrp().

8.3.10. Other Mask Bits

[Consensus Needed (Item #122d), Through end of section]:

The mask bits discussed in this section all authorize actions, that, in the absence of support for that bit mask bit, are not resolved by one of the three POSIX-derived permission bits.

It is important to note this fact because previous treatments of the bit mask have strongly suggested that each bit is either identical to a POSIX permission bit or controls a subset of one, as part of a system of controlling actions at a finer level of granularity. This seems to exclude cases like the mask bits defined in this section and in Section 8.3.11.

Where these bits are not supported, the authorization decision will be arrived at, in one of the ways listed below, with the specifics presented below as part of the discussion of that particular bit.

* The authorization can be controlled by file ownership.

* The authorization can be controlled by some boolean combination of multiple permission bits or the mask bits that correspond to those permission bits.

* The authorization can be controlled by some boolean combination file ownership and permission bits

* The particular bit does not control authorization and is only used in AUDIT and ALARM ACEs.

[Consensus Needed (Item #102b)]: The authorization information presented here is based on the only known implementation of each of the specified bits. Facilities will be provided to allow the specifics to be derived as part of mask support discovery.

ACE4_SACL_ACCESS (for all object types)

* Can only appear in AUDIT and ALARM ACEs

* If it appears in other ACEs, rejected with error NFS4ERR_INVAL.

Operations affected:
GETATTR/SETATTR of Sacl attribute

Discussion:
Triggers AUDIT/ALARM when the specified principal attempts to interrogate or change the SACL attribute

The success or failure refers to whether the change to the Sacl was successful.

Operation affected:
GETATTR/SETATTR of Acl attribute.

Discussion:
Triggers AUDIT/ALARM when the specified principal attempts to interrogate or change the Acl attribute when it contains an AUDIT or ALARM ACE (If this ACE mask bit is present, such ACEs must exist.

The success or failure refers to whether the fetch of the ACL was successful.

ACE4_SYNCHRONIZE

Operation(s) affected:
NONE

Discussion:
Permission to use the file object as a synchronization primitive for interprocess communication. This permission is not enforced or interpreted by the NFSv4.1 server on behalf of the client.

Typically, the ACE4_SYNCHRONIZE permission is only meaningful on local file systems, i.e., file systems not accessed via NFSv4.1. The reason that the permission bit exists is that some operating environments, such as Windows, use ACE4_SYNCHRONIZE.

For example, if a client copies a file that has ACE4_SYNCHRONIZE set from a local file system to an NFSv4.1 server, and then later copies the file from the NFSv4.1 server to a local file system, it is likely that if ACE4_SYNCHRONIZE was set in the original file, the client will want it set in the second copy. The first copy will not have the permission set unless the NFSv4.1 server has the means to set the ACE4_SYNCHRONIZE bit. The second copy will not have the permission set unless the NFSv4.1 server has the means to retrieve the ACE4_SYNCHRONIZE bit.

Implementation Information:

In the only known implementation of this mask bit, its effective setting, for ACEs for OWNER@, GROUP@, and EVERYONE@, is derived by oring the three POSIX-derived privilege bits for that who value.

ACE4_DELETE (For all types of objects)

Operation(s) affected:
REMOVE

Discussion:
The action of deleting the associated file or directory.

See Section 8.3.20 for information on how ACE4_DELETE and ACE4_DELETE_CHILD are to interact.

[Consensus Needed (Item #62c)]: This approach is in support of the handling of deletion when the directory’s "sticky" bit is off.

ACE4_READ_ACL (For all types of objects)

Operation(s) affected:
GETATTR (of sacl, dacl)

Discussion:
The action of interrogating the ACL of the associated file or directory.

ACE4_READ_ATTRIBUTES (for all object types)

Operation(s) affected:
GETATTR of file system object attributes

VERIFY

NVERIFY

READDIR

Discussion:
The action of reading basic attributes (non-ACLs) of a file. On a UNIX system, such basic attributes can be thought of as the stat-level attributes. Allowing this access mask bit would mean that the entity can execute "ls -l" and stat.

[Author Aside (Items #11a, #112a)]: It is not clear that existing clients are prepared to deal with denial of authorization for such operations. As a result, if this mask bit is retained, there need to be a way for clients to know, in advance that such behavior is possible before proceeding to use the file system.

If a READDIR operation requests attributes, this mask needs to be allowed for the READDIR to succeed.

[Author Aside (Items #11a, #112a)]: The suggestion above needs to be looked at closely. Even if it is justifiable to prevent access to attribute values, failing the entire READDIR seems unduly harsh. Returning an empty attribute set would be preferable as there seems no justification for making it impossible to get attributes for other files in the directory simply because a single one is so constrained.

ACE4_WRITE_RETENTION (for all object types)

Operation(s) affected:
SETATTR of retention_set, retentevt_set.

Discussion:
The action of modifying the durations for event and non-event-based retention. Also includes enabling event and non-event-based retention.

[Previous Treatment]: A server MAY behave such that setting ACE4_WRITE_ATTRIBUTES allows ACE4_WRITE_RETENTION.

[Author Aside]: The use of "MAY" here ignores the potential for harm which unexpected modification of the associated attributes might cause for security/compliance.

[Consensus Needed (Items #10a, #11a)]: If this mask bit is not supported, the specified action would be controlled by ACE4_WRITE_ATTRIBUTES .

ACE4_WRITE_RETENTION_HOLD (for all object types)

Operation(s) affected:
SETATTR of retention_hold.

Discussion:
The action of modifying the administration retention holds.

[Author Aside]: It seems that if multiple bits are to be defined for this and ACE4_WRITE_RETENTION, there would be a need to further restrict authorization for the modification of administrative retention holds.

[Previous Treatment]: A server MAY map ACE4_WRITE_ATTRIBUTES to ACE_WRITE_RETENTION_HOLD.

[Author Aside]: The use of "MAY" here ignores the potential for harm which unexpected modification of the associated attributes might cause for security/compliance.

[Consensus Needed (Items #10a, #11a)]: If this mask bit is not supported, the specified action would be controlled by ACE4_WRITE_ATTRIBUTES.

8.3.11. Possible Uses of Additional Mask Bits

The mask bits discusssed in this section all have definitions in existing specifications, but, so far, no substantive support for them has been found. As a result, any discussion of these is tentative and the working group will need to adjust appropriately when implementations are found or it is concluded that no such implementations exist.

Information about implementations of these mask bits are not yet available for various reasons:

* In the case of ACE4_READ_ATTRIBUTES, it is because the implementation investigated, while it does accept this bit in ACLs and returns it as set or not appropriately it never uses this bit to control authorization decisions

  [Author Aside (Item #112a)]: It does not seem likely that client-side software would be prepared to deal effectively with a file which could be looked up but whose attributes cannot be obtained. As a result, it appears unlikely that more substantive implementations will exist.

* In the cases of ACE4_WRITE_RETENTION and ACE4_WRITE_RETENTION, it is because the implementation investigated does not contain support for the specific OPTIONAL attributes affected.

[Author Aside (Items #10a, #11a)]: It does not appear that the descriptions of the mask bits appearing below were fully thought out before being incorporated as part of Proposed Standards. For specific cases, asides below will provide more detail. In any case, the working group will need to think about replacements while avoiding compatibility issues for any existing implementations. Any replacement would need to:

1. If separate bits for these are to be maintained and the server is allowed to refer to a coarser-grained bit when the finer-rained bit is supported, the client needs a way to determine, in advance, whether this will happen.

2. There needs to be some recognition of the fact that a retention hold, by its nature needs to be protected from arbitrary disablement as is likely to happen if the privileges to modify these or to change authorization for their change, is identical to privileges to establish them.

8.3.12. ACE Mask Bit Extension

Standards-track documents which define NFSv4 protocol extensions, as provided for in [RFC8178], can extend the set of ACE mask bits. The definition of a new mask bit needs to provide the following information:

* The set of actions affected by the new mask bit.

* The name of the new mask bit together with a previously unused mask bit value.

* Information about the way that support (or not) for the new mask bit is to be determined.

* Information about how the actions affected by this bit are dealt with when this new bit is not fully supported.

  This includes situations in which this mask bit has to have the same state some other mask bit, i.e. it is a finer-grained variant of that other bit.

  In addition case in which the new bit is not a finer-grained variant of an existing bit need to be addressed by specifying a boolean combination of existing mask bits or of conditions such as being the owner of the file or within the owning group of the file.

8.3.13. Reporting of Mask Bit Support and Characteristics

For the reasons summarized in the list below, it was necessary to provide a means of reporting information about the sets of bit masks supported, how actions controlled normally controlled by these bits are dealt with when these bits are not supported and how bits these relate to the efforts server make to coordinate the values of mode and acl-related attributes. Those means involve various aspects of the ACL_Choice attribute that are explained later.

* While previous specifications assume that all mask bits are finer-grained variants of the mask bits defined in Section 8.3.5, once one looks at all of the defined mask bits, it is clear that this cannot be the case.

  Even if that were the case, there is no way that the necessary mapping could be deduced based on the existing specifications.

* It appears that a considerable set of mask bits are finer-grained variants of file ownership.

  While the mask bits listed in Section 8.3.9 appear likely to be dealt with in this way, the lack of any explicit statement to this effect means that servers might be encountered for which it is not true.

* While it appears likely that the mask bits defined in Section 8.3.6 are treated as finer-grained variants of the write privilege, the specifications do not require this. This make it necessary that the client is aware of situations in which it is not the case.

  In addition, it is not made clear how the state of these mask bits are dealt with in computing mode.

* There is a large set of mask bits defined in Section 8.3.10 for which it is not clear exactly how the actions controlled are to be determined when the mask bits is not supported or where it is supported but is not referenced by an ACL.

* For most mask bits, existing specifications provide no way to determine whether the actions controlled are to be allowed or denied when ACEs are constructed from mode bits (e.g. when the mode is changed). Although reasonable guesses can be made for mask bits defined in Sections 8.3.6, 8.3.8, and 8.3.9, for those defined in Section 8.3.10 considerable variability is allowed and is most likely necessary.

The information necessary for the client to understand the server's mask bit handling is divided into two data items within the ACL_Choice attribute:

* The basic information as to the mask bits supported and associated defaults in the form of an ace4maskinfo, as described in Section 8.3.15

  This includes the mask bis defined in Sections 8.3.6, 8.3.8, and 8.3.9 if their handling is as described in the section that defined them. In addition mask bits defined in Section 8.3.10 can be included if their handling is not affected by other mask bits or by file ownership.

  This information is found within the ACL_Choice data item array at index ACC4IN_INFMB.

* More detailed handling information regarding mask bits whose handling cannot be satisfactorily described as above, in the form of an array of ombr4words, described in Section 8.3.16.

  This information is found within the ACL_Choice data item array at index ACC4IN_ODDMB.

8.3.14. Mask Bit Description Using ombr4words

[Consensus needed (Item #126c), through end of section]: The ombr4word data type described below has two uses:"

* For mask bits described within an ACC4IN_ODDMB data item, the handling information for the mask bit is encoded by the server directly and is available to the client in that specific form.

  This information is used to build structures used when setting modes and in computing modes corresponding to ACLs as described in Section 8.3.16.

* For mask bits described within an ACC4_INFMB, the necessary information is made available using a more compact encoding that relies on the fact that most servers have implemented mask bits in line with the original expectations for the handling of varying mask bit granularities as modified in the updated specification of ACLs.

  In order to support setting of the mode attribute and computation of the mode resulting from setting an ACL, the information in the ACC4_INFMB data item is translated to the data structures described below using the methods explained in Section 8.3.15

The data structures used in coordinating modes and ACLs are the following:

* The ACE masks used in arriving at a set of ACEs corresponding to a mode being set are represented by an acemak4 array indexed by a combination of the three POSIX privilege bits (R, W, and X) combined with an ownership bit set if the ACE principal designate the owner of the file.

  Thie array is referred to as the MaskFromPrivVal array. When considered as a four-dimensional array indexed by each of constituent booleans, it is referred to as MaskFromPrivs.

  Servers need to maintain two such array pairs for each filesystem: One for directories and one for other file objects.

* The ACE masks used in computing a mode based on an ACL being set. The ace mask privileges necessary to justify inclusion of each privilege bit is found in MaskBitsNeeded[X[where the index X is the pf the values OMBR4UAM_{READ. WRITE, EXEC}minus one.

  Servers need to maintain two such arrays for each filesystem: One for directories and one for other file objects.

The following source code section:

* Provides a basic frame for dealing with ACE masks in c++-oriented pseudo-code

* Illustrates how these coordinating data structures will be referred to in c++-oriented pseudocode used in the sections below.

```
<CODE BEGINS>
struct AceMask {
    typedef uint8       Ix;
    typedef uint32_t    Word;
    static const int    Limit = 32;
    enum class Values {         // Enum for the defined ACE mask
    };                          // with name like READ_DATA, etc.
    static const Word  DefinedWr;   // Consists of bits defined as
                                // subdivisions of the write
                                // privilege
    static const Word  DefinedRx;   // Consists of bits defined as
                                // subdivisions of the read and
                                // execute privileges
    static const Word  DefinedOwn;   // Consists of bits defined as
                                // subdivisions of the privileges
    static const Word  DefinedOth;   // Consists of bits defined as
```

```
                                    // having no determinable connection
                                    // a set of POSIX privilege bits or
                                    // ownership.
   };

   class TypeSet {
       enum class PrivIx {
           Read = 0,
           Start = 0,
           Write = 1,
           Exec = 2,
           Count = 3};
       enum class PrivVals = {
           Start = 0,
           Read = 1,
           Write = 2,
           Exec = 4,
           Own = 8,
           Count = 16};
       typedef uint32_t ace4mask;
       AceMask::Word BitsNeeded[PrivIx::Count};
       AceMask::Word MaskFromPrivVal[PrivVals::Count];
       PrivVals FormPrivVal(bool r, bool w, bool x, bool own) {
           int pv = 0;
           if (r) {
               v += (int) PrivVals::Read;
           }
           if (w) {
               v += (int) PrivVals:Write;
           }
           if (x) {
               v += (int) PrivVals::Exec;
           }
           if (own) {
               v += (int) PrivVals::Own;
           }
           return (PrivVals) v;
       };
       ace4mask & MaskRef(bool r, bool w, bool, x, bool own) {
          return MaskFromPrivVal[FormPrivVal (r, w, x, own]);
       }
   class FileSys {
       TypeSet Dirs;
       TypeSet Others;
   };
   <CODE ENDS>
```

```
<CODE BEGINS>
typedef uint32 ombr4word;

/*
 * Fields in an ombr4word including bit number, object type,
 * supported bit, needed bit, mode-computation-control, UNIX-ACL-map,
 * and mode-set-mask.
 */
const OMBR4_BNUM_SHIFT  =   0;
const OMBR4_BNUM_MASK   =   0x1f;
const OMBR4_OTYP_SHIFT  =   5;
const OMBR4_OTYP_MASK   =   0X60;
const OMBR4_SUPP_SHIFT  =   7;
const OMBR4_SUPP_MASK   =   0X80;
const OMBR4_NEED_SHIFT  =   8;
const OMBR4_SUPP_MASK   =   0X1000;
const OMBR4_MCC_SHIFT   =   9;
const OMBR4_MCC_MASK    =   0x600;
const OMBR4_UAM_SHIFT   =   11;
const OMBR4_UAM_MASK    =   0x7000
const OMBR4_MSM_SHIFT   =   16;
const OMBR4_MSM_MASK    =   0xfff00000;

/*
 * Values within the otype field
 */
const OMBR4OT_DIR       =   1;
const OMBR4OT_NDIR      =   2;
const OMBR4OT_BOTH      =   3;

/*
 * Values within the mode-computation-control field
 */
const OMBR4MCC_NONE     =   0;
const OMBR4MCC_READ     =   1;
const OMBR4MCC_WRITE    =   2;
const OMBR4MCC_EXEC     =   3;

/*
 * Values within the UNIX-ACL-map field
 */
const OMBR4UAM_NEVER    =   0;
const OMBR4UAM_READ     =   1;
const OMBR4UAM_WRITE    =   2;
const OMBR4UAM_EXEC     =   3;
const OMBR4UAM_ALWAYS   =   4;
const OMBR4UAM_OWNER    =   8;
```

```
   /*
    * Values of shifts used in forming the mode-set mask field.
    */
   const OMBR4MSH_RBIT     =   1;
   const OMBR4MSH_WBIT     =   2;
   const OMBR4MSH_XBIT     =   4;
   const OMBR4MSH_OWN      =   8;

   /*
     * Values of the mode-set mask field to reflect dependence on a
     * single one of the indices used in accessing the sixteen-bit
     * integer as a four-dimensional boolean array.
     */
   const OMBR4MSM_RBIT     =   0xaaaa;
   const OMBR4MSM_WBIT     =   0xcccc;
   const OMBR4MSM_XBIT     =   0xf0f0;
   const OMBR4MSM_OWN      =   0xff00;
   <CODE ENDS>
```

In order to facilitate reference to the above data in c++-oriented pseudocode in the sections below, a class Ombr4 will be defined. It will contain a single 32-bit data item Word of type Value (uint32_t) together with appropriate Get and Set methods.

In addition, for each Xyz within the set including Bnum, Otyp, Supp, Mcc, Uam, and Msm, there will need to be:

* Static const integers XyzShift and XyzMask.

* Typedefs for each field named XyzType.

  SuppType and NeedType will be bool while MsmType wlll be uint16_t and BnumType will be uint8_t.

  For Bnum, Otyp, Mcc, and Uam, XyzType will be a class enum using the values defined in the XDR above.

* There will be XyzExtract returning XyzType with a Value parameter.

* There will be an XyzInsert with a Value & first parameter and an XyzType second parameter.

In addition, for Msm there will need to be an inner class Ombr4::Msm including the listed below.

* A named enum Ombr4::Msm:Shift with values derived from OMBR4MSH_{?BIT,OWN}.

* A method MaskFromShift with a Msm::Shift parameter returning an MsmType.

* A named enum Ombr4::Msm:Privmask with values derived from OMBR4MSM_{?BIT,OWN}.

* A method MaskFromShift with a Msm::Shift parameter returning a Msm::PrivMask.

8.3.15. Basic ACE Mask Support Info

[Consensus needed (Item #126d), through end of section]: This section describes the ACE mask support information that is needed to needed to fully specify the needs of client and server for information relevant to the handling of ACE mask bits

This information is available in the form of an ace4maskinfo, as described below:

```
<CODE BEGINS>
struct ace4masksupp {
        acemask4        acms_valid;
        acemask4        acms_separate;
        acemask4        acms_needed;
};
struct ace4maskinfo {
        ace4masksupp    acmi_dir;
        ace4masksupp    acmi_ofo;
        acemask4        acmi_defall;
        acemask4        acmi_defown;
};
<CODE ENDS>
```

Because the mask granularity might be different for files and directories, an ace4maskinfo contains two ace4masksupp structure: one for directories and one for other file objects. Each is composed of the following fields.

* acms_valid is a mask consisting valid ACE mask bit, i.e., those which may be specified with an ACE mask field and acted upon, neither ignored or rejected as invalid.

* acms_separate is a mask consisting of bits that are separately specifiable so that it and other bits derived from the same coarser-granularity mask bit can be set separately, allowing the corresponding set of actions to be either allow or denied independently.

* acms_needed in a mask consisting of all the mask bits that need to be set to justify turning on the corresponding privilege bit when computing the mode consistent with a newly set ACL.

  It might be supposed that this set could be derived from the intersection of acms_valid and acms_separate, suitably restricted to the union of the associated core mask bits and one on the sets defined in Sections 8.3.6 or 8.3.8. While this approach deals with the possible splitting of write permissions into appending to a file and overwriting existing data, there are other case n which it is not suitable/

  For mask bit such as ACE4_WRITE_NAMED_ATTRS and ACE4_WRITE_EXT_ATTRS, it is often the case that the non-allowance f these mask bits is not sufficient to treat the file as read-only. This is likely to occur when filesystems are upgraded to support, often on a tentative basis, either named or extended attributes.

It is important to note that the setting of each mask bit in acms_valid and acms_separate fundamentally independent. This is the case even though there are often practical considerations that make it inadvisable to include a mask bit in acms_separate without including it in acms_valid. In is important to understand how such mask bits interact with acmi_defall and acmi_defown.

An ace4maskinfo structure consists of the following elements:

* The acmi_dir and acmi_ofo structures provide mask bit support information for directories and other file objects respectively.

* acmi_defall indicates the mask bits for actions that are allowed by default, whenever not explicitly denied.

  This is useful for environments in which it is typical to grant such rights unless they are explicitly denied, as might often be necessary in POSIX-based environments in dealing with the ACE4_READ_ATTRIBUTES mask bit.

* acmi_defown indicates the mask bits for actions that are allowed by default when performed by the owner of the file.

  Mask bits set in this field are of special significance when the special "who" value OWNER_RIGHTS@ is supported, as described in Section 8.4.3

Even though there are only single ace4masks for acmi_defown and acmi_defall for all object types, there are ways that there can be different defaults for directories and other file objects. The is because a mask bit in one of the default masks that corresponds to a mask bit that is zero in both the acms_valid and acms_separate mask words is effectively ignored.

The information necessary to support the computation of modes based on ACLs, as described in Section 10.5 can be derived as described below.

This set of information, the set of mode-computation control bits, consists of three ACE masks for each of the read, write, and execute privilege bits. There needs to be one such set for directories and one for non- directory objects with the set for directories derived from acmi_dir and the one for other objects derived from acmi_ofo.

Initially both sets of mask bits ae filled in with values that reflect the mapping between privilege bits and the corresponding ACE mask bit so that:

* The bit mask for read contains the mask bits for ACE4_READ_DATA/ ACE4_LIST_DIRECTORY.

  In addition, the bit mask including other supported bit defined in Section 8.3.8

* The bit mask for write contains the mask bits for ACE4_WRITE_DATA/ ACE4_ADD_FILE.

  In addition, the bit mask including other supported bit defined in Section 8.3.6

* The bit mask for execute contains the mask bits for ACE4_EXECUTE.

  In addition, the bit mask including other supported bits defined in Section 8.3.8

The information necessary to support the generations of ACEs corresponding to modes, as described in Section 10.7.

This set of information, the set of mode-set mask support bits, consists consist of thirty-two unsigned 16-bit integers, one for each potential ACE mask bit. As with the other sorts of information that needs to be determined, there needs to be one such set for directories and one for non- directory objects.

Once the appropriate integer is selected, it is used essentially as a linearized four-dimensional boolean array, with the four indices consisting of the read, write and execute privilege bits and whether the current ACE is for the owner of the file, as with the MSM field of an ombr4word described in Section 8.3.13.

Initially each of these consist of 32 16-bit zero words. They are then updated to reflect the proper handling of mask bits defined in Section 8.3.5 as follows.

* The word at position ACE4_READ_DATA/ACE4_LIST_DIRECTORY receives the value OMBR4MSH_RBIT.

* The word at position ACE4_WRITE_DATA/ACE4_ADD_FILE receives the value OMBR4MSH_WBIT.

* The word at position ACE4_EXECUTE receives the value OMBR4MSH_XBIT.

The information necessary to support the conversion of UNIX-oriented ACE mask words (i.e., those that only consist of three valid ACE mask bis) to those appropriate to a particular file system which might support a larger set of mask bits.

This set of information, the set of UNIX-ACL-expansion mask consists of two sets of three ACE masks for each the read, write, and execute privilege bits. One is used for distinguishing access by the file owner from that for users other than OWNER@. As with the case of mode-computation control bits, there needs to be one such set for directories and one for non- directory objects.

For clients, the information, where needed, is obtained as follows:

* When the ACL_Choice attribute is supported, the information described above can be derived from the contents of that attribute as described in Section 8.3.17.

* When the ACL_Choice attribute is not supported, the information described is replaced by various defaults described in Section 8.3.18. These defaults may be inaccurate in a number of respects but they are designed so that UNIX ACLs can be translate into those appropriate for servers who support finer-grained authorization. To allow that to happen, servers need to follow the recommendations in Section 6.3. While servers who do not follow these recommendations will often be, formally speaking, spec-compliant, it needs to be understood that such servers may behave in unexpected manners when interacting with clients who have no way to find out where exactly the problem lies.

The pseudocode below provides a description of how the arrays described above can be derived:

```
<CODE BEGINS>
//
// FileSysPlus -- Extends FileSys to accommodate the possibility of
//                dealing with the data item ACC4IN_INFMB.
//
// Documents how the fields used in setting modes and computing modes
// are to be derived from the data in the ACC4IN_INFMB data item.
// The possibility that this is not present by allowing the
// MaskInfo pointer InfoAddr to have the value NULL.
//

class FileSysPlus : FileSys {

    //
    // Corresponds to the XDR structure ace4masksupp. Note that
    // byte-swapping is not explicitly dealt with.
    //

    struct MaskSupp {
       ace4mask     Valid;
       ace4mask     Separate;
       ace4mask     Needed;
    };

    //
    // Corresponds to the XDR structure ace4maskinfo. Note that
    // byte-swapping is not explicitly dealt with.
    //

    struct MaskInfo {
       MaskSupp    Dirs;
       MaskSupp    Others;
       ace4mask    DefAll;
       ace4mask    DefOwn;
    };

    //
    // Address of the associated MaskInfo obtained from
    // ACC4IN_INFMB data item, if any.
    //

    MaskInfo * InfoAddr;

    //
    // Fill in the bits-needed array for either directories or
```

```
       // non-directories.
       //
       bool InfFillBitsNeeded(bool forDir); {
           MaskSupp *  msp = forDir ? &InfoAddr->Dirs
                                    : &InfoAddr->Dirs;
           ace4mask *  bnp = forDir ? Dir->BitsNeeded
                                    : Others->BitsNeeded;
           int         ix;
           ace4mask    v;
           for (ix = (int) PrivIx::Start;
                ix < (int) PrivIx::Count;
                ix++, bnp++) {
               v = PrivToCore((PrivIx) ix);
               if (ix == (int) PrivIx::Write) {
                  v |= (ace4mask) AceMask::DefinedWr;
               }
               else {
                  v |= (ace4mask) AceMMask::DefinedRx:
               }

               ace4mask n = (msp->Valid | msp->Separate) &
                            msp->Needed;
               *bnp = n & v;
            };
       };

       //
       // Fill in the MaskFromPrivVal array for either directories or
       // non-directories.
       //

       bool InfFillMaskFromPrivVal(bool forDir); {
           MaskSupp * msp = forDir ? &InfoAddr->Dirs
                                   : &InfoAddr->Dirs;
           ace4mask * pvp = forDir ? Dir->MaskFromPrivVal
           g1
           : Others->MaskFromPrivVal;
           int         ix;
           ace4mask    v;

           for (ix = (int) PrivVals::Start;
                ix < (int) PrivVals::Count;
                ix++, pvp++) {
              bool r = (ix & PrivVals::Read;
              bool w = (ix & PrivVals::Write;
              bool x = (ix & PrivVals::Exec;
              bool own = (ix & PrivVals::Own;
       };
```

```
        //
        // Fill in both MaskFromPrivVal arrays if the info is available
        //
        // The value returned is a success indication with the case in
        // which the info is unavailable treated as a (nominal) success
        // since nothing went wrong.
        //

        bool InfFillMaskFromPrivVal(); {
            if (InfoAddr == NULL) {
                return (true);
            };
            return (InfFillMaskFromPrivVal(true) &&
                    InfFillMaskFromPrivVal(false));
        };

        //
        // Fill in both BitsNeeded arrays if the info is available
        //
        // The value returned is a success indication with the case in
        // which the info is unavailable treated as a (nominal) success
        // since nothing went wrong.
        //

        bool InfFillBitsNeeded() {
            if (InfoAddr == NULL) {
                return (true);
            };
            return (InfFillBitsNeeded(true) &&
                    InfFillBitsNeeded(false));
        };
    <CODE ENDS>
```

8.3.16. Description of General Mask Bit Handling

[Consensus needed (Item #126e), through end of section]: This section discusses the provision of a full description of a server's handling of ACE mask bits as a array of ombr4words indexed by bit position. Although it is possible to specify the handing of all mask bits in this way, it is more common to express the handling of those bits whose handing is typical as described in Section 8.3.15 while using this method for those bits, if any, that cannot be accommodated in that way.

The reporting described below might turn out to be more general than is necessary. However, we have strived to make it as general as possible because existing specifications have not put any limits on the handling of most mask bits. Any case in which the result is not

usable for appropriately mapping a client's ACLs to those of the server will be dealt with in Section 12.5. Because existing specifications allowed a large range of potential behaviors in handling of mask bits, most indications of the need to avoid certain approaches are addressed by using what we term a "residual SHOULD" where it is made clear that only valid reason to bypass the recommendation is an implementor's reliance on a previous valid specification with the consequences of any such bypass on client functioning being made clear.

Reporting is necessary for all defined mask bits except those for which the action to be controlled can never occur. While the case of ACE4_SYNCHRONIZE is not included among these, the following cases exclusions are sometimes necessary:

* ACE4_READ_NAMED_ATTRIBUTES and ACE4_WRITE_NAMED_ATTRIBUTES are to be excluded on file systems in which named attributes are not supported/

* ACE4_WRITE_RETENTION and ACE4_WRITE_RETENTION_HOLD are to be excluded on file systems in which the retention attributes are not supported.

Reporting of mask bit characteristics depends initially on the subsection in which the mask bit is defined. In most cases, this takes care of the issue, but, for each of the groups listed below, it is possible that a later default method will be needed.

* For mask bits defined in Section 8.3.5, no reporting is necessary, since support for these mask bits is REQUIRED

  Even though these are the only mask bits UNIX-oriented clients deal with, such clients need to be able to interact with servers that do support other mask bits, as described in Section 8.3.19.

* For mask bits defined in Section 8.3.6, support normally is indicated by setting the bits ACC4BN_SEPFWX, ACC4BN_SEPAFD, ACC4BN_SEPDE, and ACC4BN_SEPWNA in the ACL_Choice attribute

  When the mask bit in question is referred to in computing the mode corresponding to an ACL or is not derived from the value of write privilege bit, when a mode is set, this mask bit is reported using the default method, whether the mask bit is supported or not.

* For mask bits defined in Section 8.3.9, support normally is indicated by setting the mask bit within the acemask4 associated with data item ACC4IN_OWNMB in the ACL_Choice attribute.

When the mask bit in question is referred to in computing the mode corresponding to an ACL or is not derived from the mode in any other way but by setting the bit in the ACE for OWNER@ this mask bit is reported using the default method, whether the mask bit is supported or not.

* For mask bits defined in Section 8.3.10, the default method is always used, whether the mask bit is supported or not.

When the default method is to be used for a given mask bit, either one or two sets of ombr4words are created and added to the data item. One can be used if the handling of the mask bit is the same for directory and non-directory objects, with use of two being available where there is any difference. Normally, each set consists of only a single word is necessary but two can be used to possible conflicting uses

The handling for various mask bits is represented by ombr4words, as described in Section 8.3.14.

Whether one or two words is used, the various fields defined above are filled in as described below:

1: The BNUM field gets the bit number of the mask bit being described in the form of the number of bit to left-shift the value one to get the value of the mask bit

2: The OTYP field is set to either OMBR4OT_BOTH, if a single word is being filled in applying to all objects, or to OMBR4OT_DIR or OMBR4OT_NDIR for each word depending on the objects being described.

3: The SUPP field is filled in with a one or zero depending on whether the mask bit is supported as defined below.

In this context we define a mask bit as "supported" only if the actions controlled can be allowed or disallowed, independently of the handing of other actions.

4: Fill in the MCC field based the kind of effect the allowance of a particular mask has in computing a mode based on ACL:

The value OMBR4MCC_NONE indicate that the setting of the bit has no effect on the mode being computed.

The value OMBR4MCC_READ indicates that when the action is allowed, the associated read privilege bit s is turned on. on.

The value OMBR4MCC_WRITE indicates that when the action is allowed, the associated write privilege bit is turned on. on.

The value OMBR4MCC_EXEC indicates that when the action is allowed, the associated exec privilege bit is turned on. on.

5: Fill in the UAM field to indicate the appropriate setting to indicate how the bit is to be dealt with in translating a UNIX ACL:

The value OMBR4UAM_ALWAYS indicates that the bit's actions are always to be allowed in the resulting ACL.

The value OMBR4UAM_NEVER indicates that the bit's actions are never to be allowed in the resulting ACL.

The value OMBR4UAM_READ indicates that the bit's actions are to be allowed in the resulting ACL if read privilege bit is set.

The value OMBR4UAM_WRITE indicates that the bit's actions are to be allowed in the resulting ACL if the write privilege bit is set.

The value OMBR4UAM_OWN is ored into the value to indicates that the bit's actions are to be allowed in the resulting ACL iff the user is the owner of the file.

6: Fill in the MSM field to indicate the appropriate handling of this mask bit when setting the mode attribute. The value is a sixteen-bit value to indicate whether the action is to be allowed based on the setting of the three privilege bits and whether the entry being generate is for the owner of the file.

Each of the bits within this field describes whether action specified is to be allowed. The bit offset for each combination of privilege bits and file ownership is determined by summing the following values:

* OMBR4MSH_RBIT if the read privilege bit is set.

* OMBR4MSH_WBIT if the write privilege bit is set.

* OMBR4MSH_XBIT if the execute privilege bit is set.

* OMBR4MSH_OWN if the ACE being built is for the owner of file.

The pseudo-code below provides a description of how the arrays described above can be derived from the ombr4words within the

```
<CODE BEGINS>
//
// FileSysPlusOddOnes -- Extends FileSysPlus to accommodate the
//                       possibility of dealing with the data item
//                       ACC4IN_ODDMB.
//
// Documents how the fields used in setting modes and computing modes
// are to be derived from the data in the ACC4IN_ODDMB data item.
// The possibility that this is not present is addressed by allowing
// the OddInfo  pointer OddAddr to have the value NULL or the
// OddCount to be zero.
//

struct FileSysPlusOddOnes : FileSysPlus {

    uint32_t        OddCount;
    Ombr4::Value *  OddInfo;

    Ombr4::Value *  FindInfo(bool dir, AceMask::Ix ix) {
        if (OddInfo ==  NULL || OddCount == 0) {
            return (NULL);
        };
        AceMask::Ix     CurIx;
        Ombr4::Value *  CurPtr = Oddinfo;
        for (CurIx = 0;  CurIx < OddCount; CurIc++. CurPtr++) {
            Ombr4::Value    Cur = *CurPtr;

            if (Ombr4::ExtractBnumm(Cur) != ix) {
                continue;
            };

            Ombr4::OtypType t = Ombr4::ExtractOtyp(Cur);
            if (t == Ombr4::OtypType::Both) {
                return CurPtr;
            };
            Ombr4::OtypType t2 = dir ? Ombr4::OtypType::Dir
                                     : Ombr4::OtypType::Ndir;
            if (t == t2) {
               return CurPtr;
            };
        };
        return (NULL);
    };
    void ApplyOddOne(TypeSet *tsp, Ombr4::Value v) {
        Ombr4::BnumType bn = Ombr4::ExtractBnumm(v);
        AceMask::Word   w = 1 << bn;
        bool            supp = ExtractSupp(w)
        bool            needed = ExtractNeeded(w);
```

```
            Ombr4::MsmType  msm = ExtractMsm(w);
            bool            ok = true;
            Typeset::PrivIx pix;

            switch (mt) {

                case Ombr4::MccType::Write:
                    pix = TypeSetPrivx::Write;
                    break;

                case Ombr4::MccType::Read:
                    pix = TypeSetPrivx::Read;
                    break;

                case Ombr4::MccType::Exec:
                    pix = TypeSetPrivx::Exec;
                    break;

                default:
                    ok = false;
            };
            if (ok && supp && needed) {
                BitsNeeded[pix] |= w;
            }
            else {
                BitsNeeded[pix] &= ~ w;
            }

            uint8_t pvx;
            pvm = 1;

            for (pvx = (uint8_t) Typeset::PrivVals::Start;
                 pvx < (uint8_t) Typeset::PrivVals::Count;
                 pvx++. pvm <<= 1) {

                if (pvm & msm {
                    MaskFromPrivVal[pvx] |= w;
                }
                else {
                    MaskFromPrivVal[pvx] &= ~w;
                }
            }
        }

        void ApplyOdd(bool dir) {
            TypeSet *       tsp = dir ? & Dirs : & Others;
            AceMask::Ix     ix;
            Ombr4::Value *  p;
```

```
            for (ix = 0; ix < AceMask::Limit; ix++) {
               p = FindInfo(dir, ix);
               if (p != NULL) {
                   ApplyOddOne(tsp, *p);
               };
            };
        };
        void ApplyOdd()
            if (OddInfo ==  NULL || OddCount == 0) {
                return (NULL);
            };
            ApplyOdd(true);
            ApplyOdd(false)
        };
```

<CODE ENDS>

8.3.17. ACE Mask Support Discovery

This section describes how to determine the ACE mask support information once it is initialized as described in Section 8.3.15. On the client, the information is updated using the contents of the ACL_Choice attribute as described below. On the server, the values presented in the ACL_Choice attribute will match the ACE mask support used here. The intention is that the client and server have the same ACE mask support information and are aware of each other’s handling of the related processes.

Of the ACE mask bits defined above and those added later by extensions as described in Section 8.3.12, support for most is OPTIONAL. The only exceptions are for those discussed in Section 8.3.5, which need to be supported whenever ACL-related attributes are supported. The ACE mask support information when initialized as described in Section 8.3.15 reflects support only for those mask bits.

Information regarding most of the supported mask bits defined in Section 8.3.6 is based on determining the sets of mask bits to add to for both directory and non-directory objects, as described below using the ACL_Choice flag bits:

* When the flag bit ACC4BN_SEPFWX is set, ACE4_APPEND_DATA is added to the mask bits for non-directory objects.

* When the flag bit ACC4BN_SEPDE is set, ACE4_DELETE_CHILD is added to the mask bits for directories.

* When the flag bit ACC4BN_SEPAFD is set, ACE4_ADD_SUBDIRECTORY is added to the mask bits for directories.

* When the flag bit ACC4BN_NAD is set, ACE4_WRITE_NAMED_ATTRIBUTES is added to both masks.

Once these masks have been determined, each bit set in one of the masks causes the following changes in the ACE mask support information:

* The mask bit is added to the corresponding mask of supported ACE bits.

* Within the entry within the set of mode-computation control bits devoted to the write privilege, the bit is added to the set of corresponding ACE mask bits.

* Similar changes are made to the sets of UNIX-ACL-expansion bits. The same changes are made to masks for owners and for non-owner.

* The set of mode-set mask support bits for the mask bit in question receives the value OMBR4MSH_WBIT.

At the point, the bits set in the mask associated with ACL_Choice data item ACC4IN_OWNMB are applied to data for both directory and non-directory objects. The following changes are made for each set bit.

* The bit is added to the set of supported ACE mask bits.

* No changes are made to the set of mode-computation control bits or the set of UNIX-ACL-expansion bits.

* The entry for the mask bit in set of mode-set mask support bits receives the value OMBR4MSH_OBIT.

After processing the mask bits whose role in the ACL semantics are clear, the ombr4words in ACL_Choice data item ACC4IN_ODDMB are applied to deal, with mask bits defined in Section 8.3.10 and those defined in Sections 8.3.6 and 8.3.9 whose semantic is any way atypical. Successive ombr4words are obtained from that item and applied to the data for directories, non-directory-objects or both as specified in the OTYP field. For each such words, the following changes are made:

* If the SUPP field is non-zero the set(s) of supported mask bits are updated to add the current mask bit.

* Mode computation bits are update to reflect the value of the MCM field. if that field is not OMBR4MCC_NONE, then the current mask bit is added to the specific mask denoted by that field current mask bit

* UNIX-ACL map control bits are updated as specified by the UAM field. Normally the sets for both owner and non-owner entries are specified but if OMBR4UAM_OWNER is set, only the owner entries are modified. Form this point the handling depends on the UAM field with this it removed.

  If the value is OMBR4UAM_READ, the current mask is ored into the mask(s) associated with the read privilege.

  If the value is OMBR4UAM_WRITE, the current mask is ored into the mask(s) associated with the write privilege.

  If the value is OMBR4UAM_EXEC, the current mask is ored into the mask(s) associated with the execute privilege.

  If the value is OMBR4UAM_ALWAYS, the current mask is ored into the masks for all of the privileges.

* The value of the MSM field is transferred to the mode-set-mask control bit for the specified mask bit

8.3.18. ACE Mask Bit Support Defaults

When the ACL_Choice attribute is not supported, default ACE mask support information is initialized based on the set of write-associated mask bits.

The set of write-associated mask bits starts with a set based on object type:

* For directories, it includes ACE4_ADD_FILE, ACE4_ADD_SUBDIRECTORY,and ACE4_DELETE_CHILD

* For non-directory objects, it includes ACE4_WRITE_DATA and ACE4_APPEND_DATA.

Then ACE4_WRITE_NAMED_ATTRIBUTES is added to both sets if named attributes are supported. Then the ACE mask support information is generated as described below.

* The set of supported mask bits are limited to those that are REQUIRED: ACE4_READ_DATA/ACE4_LIST_DIRECTORY, ACE4_WRITE_DATA/ ACE4_ADD_FILE, and ACE4_EXECUTE.

This default might be inaccurate but clients have no way to determine what extensions are implemented by the server

* Within the entry within the set of mode-computation control bits devoted to the write privilege, each write-associated bit is added to the set of corresponding ACE mask bits.

* Similar changes are made to the sets of UNIX-ACL-expansion bits. The same changes are made to masks for owners and for non-owner.

* The set of mode-set mask support bits for each mask bit within the write-associated mask bits receives the value OMBR4MSH_WBIT.

8.3.19. ACE Mask Adaptation

When, as is often the case, clients are limited to the use of the three REQUIRED ACE mask bit they need to be able to appropriately adapt the ACL they use in interacting with the server to reflect a larger set of ACE mask bits that the server is using. This can occur when ACL_Choice is supported, in which case the UNIX-ACL mapping information is derived as described in Section 8.3.17 or when it is not supported in which case that information is generated as described in Section 8.3.18.

The ACEs are processed in turn regardless of their ACE types. For each ACE, a new mask needs to be arrived at based on the following information:

* Whether ACE4_READ_DATA/ACE4_LIST_DIRECTORY is set in the original ACE mask.

* Whether ACE4_WRITE_DATA/ACE4_ADD_FILE is set in the original ACE mask.

* Whether ACE4_EXECUTE is set in the original ACE mask.

* Whether the ACE specifies OWNER@ s the who value

Based on the who value, either the owner or non-owner bits are selected.

Then, the appropriate set of masks are ored together with the masks selected based on the original ACE mask controlling which masks are selected. The resulting mask become the ACE mask for the new ACE.

8.3.20. Handling of Deletion

[Author Aside, Through end of bulleted list]: This section contains a proposal for the revision of the ACL-based authorization handling of requests to delete directory entries or to do RENANE operations. A completely new description is necessary because earlier descriptions were not appropriately coordinated with the non-ACL authorization of these action, do not clearly address the potential no-support of ACE4_DELETE, and have other flaws noted later in this section. The handling described in Section 6.2.1.3.2 of [RFC8881] arose as follows:

* Previous specifications (e.g., [RFC8881]) do not reference the role MODE4_SVTX authorizing deletion and RENAME at all in the non-ACL case. As a result the issue of coordinating ACL-based and non-ACL authorization was never addressed.

* Although previous specifications defined an ACE4_DELETE bit, it was never made clear what having such a bit was intended to accomplish. This gap was particularly important given that, unlike many ACE mask bit outside of core three, there is no way that this bit can be a finer-grained variant of some of the privilege bit, since normally (in the non-sticky-bit case), there is no additional file-based permission required for deletion.

* When [RFC8881] and its predecessors were written, the result wound up undercutting the most likely use of ACE4_DELETE, to allow a file to be made undeletable. In addition it allowed the function of ACE4_DELETE_CHILD to be undercut without any apparent good reason.

  In any case, this would need to be redone to align the ACL-based treatment with the authorization described in Section 5.3.1 of the new security document ([I-D.dnoveck-nfsv4-security]). This is apart from the need to correct the issues mentioned above.

[Author Aside]: All unannotated material within this section is to be considered part of consensus item #12c. The associated previous treatment is to be found in Appendix B.1

This section describes the handling requests of that involve deletion of a directory entry or the disappearance of directory entry as a result of RENAME.

It supersedes the previous treatment of these issues that appears in Section 6.2.1.3.2 of [RFC8881] and some predecessor specifications. Because a number of flaws that will be elaborated on later in this section. that previous description SHOULD NOT be implemented. In

this context the reliance on previous Proposed Standards is considered a valid reason to bypass the above recommendation as long as the implemented is aware that:

* That treatment makes it impossible to use denial of ACE4_DELETE to deny users permission to delete a file. This is of particular concern since it is hard to find any other motivation to define an ACE4_DELETE ACE mask bit.

* The use of MODE4_SVTX to indicate an additional requirement for authorization (using the phrase "may also require") may result in client and user confusion since it is different from what, as far as can be determined from the non-ACL handling of the sticky bit on directories.

* The allowance of deletion if (only !!) the ACE4_DELETE succeeds undercuts the normal assurance about directory write privileges and might constitute a security problem.

  This is of particular concern given the lack of a clear statement that such allowance needs to be explicit. Given the lack of clarity, it is possible that when this bit is not used or supported, the function of the write privilege bit for the directory will be totally undercut.

* The lack of any discussion of AUDIT and ALRAM ACEs makes it likely that if this approach is adopted by servers supporting those ACE types, clients are very unlikely to see expected results, especially when MODE4_SVTX is set.

It needs to be noted that when the server chooses to do this it needs to signal this choice using the ACC4BN_SVTXOLD bit when the ACL_Choice attributed is supported

* The creation of a new directory entry , as happens in RENAME is not covered and is described elsewhere in this document. That is, the directory updates require the privileges for directory modification described elsewhere and there is no reason to check ACE4_DELETE since no file is deleted. However, the deletion of files and directories as result of RENAMEs (i.e., in the rename-over case) is covered.

* The deletion of the file’s data due to other operations is dealt with separately since these actions, including a truncation to length zero, requires only ACE4_WRITE_DATA on the target file

* In cases in which a file or directory is deleted, permission checks on the directory need to be supplemented by checks of ACE4_DELETE on the entity to be deleted.

* The server MAY, when MODE4 SVTX is set on the directory allow authorization of the modification to depends on the requester being the owner of the target file or the directory, rather than on write authorization or the appropriate finer-grained ACE-mask bit, when that mask bit is supported.

* The confused state of previous specifications makes it necessary that we have extensive information about existing implementations before we close on Consensus Item #12.

Special authorization of such operations need to be specified here because of two special circumstances that mean that the authorization cannot be done as described for other operations or for which checking needs to be done for a specific permissions on either a file or a directory:

* In cases in which a file or directory is deleted, permission checks on the directory need to be supplemented by checks of ACE4_DELETE on the entity to be deleted.

* The server MAY, when MODE4 SVTX is set on the directory allow authorization of the modification to depends on the requester being the owner of the target file or the directory, rather than on write authorization or the appropriate finer-grained ACE mask bit, when that mask bit is supported.

With regard to the check made on the entity to be deleted:

* If the entity does not have an ACL, the check is unnecessary and is deemed to have succeeded.

* If the file system being operated upon does not support the ACE4_DELETE mask bit, no check is done and success is assumed.

* If the file system being operated upon does support the ACE4_DELETE mask bit, and that bit is not explicitly allowed or denied, the default is chosen by the server.

  In the case in which the default is to allow the operation that default needs to be reported when the ACL_Choice attribute is supported.

With regard to the permission check made on the directory being modified:

* The normal permission check for directory modification is done if the MODE4_SVTX bit for the directory is not set.

* The appropriate mask bit to be checked depends on the operation being performed.

  In the case of a delete, ACE4_DELETE_CHILD is checked.

  In the case of a cross-directory RENAME, ACE4_DELETE_CHILD is checked on the source directory and either ACE4_ADD_FILE or ACE4_ADD_SUBDIRECTORY (depending on the type of the object being renamed) is check on the destination directory.

  In the case of a within-directory RENAME, the server MAY treat this as a removal followed by insertion as in the cross-directory case or as only a modification to the directory, requiring ACE4_ADD_FILE only. In the former case, the server's choice to take this approach is to be reported by setting the ACC4BN_RNASDI bit when the ACL_Choice attribute is supported.

  In all of the above cases, if the selected mask bit is not supported, ACE4_ADD_FILE is checked instead.

* If the directory does not have an ACL, the write privilege for the directory is checked instead.

* If the MODE4_SVTX bit for the directory is set, the server MAY replace the normally appropriate directory check by a check as to whether the requester is the owner of the directory or of the target file.

  The potential is generate limited to the source directory in the RENAME case. The only case in which it applies to the destination directory concerns check or deletion in the rename-over case.

  When the server chooses to do this, that choice is reported by setting the ACC4BN_SVTX bit when the ACL_Choice attribute is supported.

With regard to handling of AUDIT/ALARM when these operations are done:

* When multiple checks are made, the order is up to the server.

  As a result, when one of the checks fails, the user has no guarantee that other checks typically done later are done, so that audit and alarm events for these later potential checks might not be recognized.

* When the operation succeeds, the user can assume that all check are done, so that audit and alarm events for all of the checked privileges will be recognized.

* When multiple checks are done and some succeed but others fail, the audit and alarm events associated with the failed checks will be reported as failures, For the successful checks, audit and alarm events are recognized as well but it is up to the server whether to report them as successes (because the check succeeded) or failed attempts (because the operation was not done.

* In case in which a directory check is superseded by an ownership checks (i.e., because of the presence of MODE4_SVTX), recognition of audit and alarm events proceeds as it would have in the absence of MODE4_SVTX.

8.4. ACE flag bits

The bitmask constants used for the flag field are as follows:

```
<CODE BEGINS>
const ACE4_FILE_INHERIT_ACE             = 0x00000001;
const ACE4_DIRECTORY_INHERIT_ACE        = 0x00000002;
const ACE4_NO_PROPAGATE_INHERIT_ACE     = 0x00000004;
const ACE4_INHERIT_ONLY_ACE             = 0x00000008;
const ACE4_SUCCESSFUL_ACCESS_ACE_FLAG   = 0x00000010;
const ACE4_FAILED_ACCESS_ACE_FLAG       = 0x00000020;
const ACE4_IDENTIFIER_GROUP             = 0x00000040;
const ACE4_INHERITED_ACE                = 0x00000080;
<CODE ENDS>
```

[Author Aside]: Although there are multiple distinct issues that might need to be changed, in the interest of simplifying the review, all such issues within this section will be considered part of Consensus Item #13a with a single revised treatment addressing all the issues noted.

[Previous Treatment]: A server need not support any of these flags.

[Author Aside]: It is hard to understand why such broad license is granted to the server, leaving the client to deal, without an explicit non-support indication, with 256 possible combinations of supported and unsupported flags. If there were specific issues with some flags that makes it reasonable for a server not to support them, then these need to be specifically noted. Some additional problems with the existing treatment are that many of the flags in this list are not, by their nature, capable of support by the server (e.g., ACE4_INHERITED_ACE) and that some cannot reasonably be made OPTIONAL

in general (e.g. ACE4_IDENTIFIER_GROUP for all ACEs and both ACE4_SUCCESSFUL_ACCESS_ACE_FLAG and ACE4_FAILED_ACCESS_ACE_FLAG for AUDIT and ALARM ACES). Also troubling is the lack of any statement regarding the use of unassigned bits and the consequent effect on protocol extensibility.

[Previous Treatment]: If the server supports flags that are similar to, but not exactly the same as, these flags, the implementation may define a mapping between the protocol-defined flags and the implementation-defined flags.

[Author Aside]: The above regarding, the possibility of defining a mapping between the protocol-defined flags and hypothetical implementation-defined flags might store the bits it supports, while valid, is out-of-scope and needs to be deleted.

[Previous Treatment]: For example, suppose a client tries to set an ACE with ACE4_FILE_INHERIT_ACE set but not ACE4_DIRECTORY_INHERIT_ACE. If the server does not support any form of ACL inheritance, the server should reject the request with NFS4ERR_ATTRNOTSUPP. If the server supports a single "inherit ACE" flag that applies to both files and directories, the server may reject the request (i.e., requiring the client to set both the file and directory inheritance flags). The server may also accept the request and silently turn on the ACE4_DIRECTORY_INHERIT_ACE flag.

[Author Aside]: Aside from the lack of clarity arising from use of the lower-case alternatives to RFC2119-defined keywords, what is the possible justification for accepting a request asking you do something and then, without notice to the client, do something else. I believe there is none.

Consensus Needed (Item #13a)]: Server support for many of the flags defined above is OPTIONAL, although there are constraints in some cases so that certain combination of support and non-support are not allowed, as described in Section 8.4.1

Consensus Needed (Item #13a)]: When a server which does not support all the flags bits receives a request to set an NFSv4 ACL containing an ACE with an unsupported flag bit set the server MUST reject the request with NFS4ERR_INVAL or NFS4ERR_ATTRNOTSUPP

Consensus Needed (Item #13a)]: The case of servers which do not provide support for particular flag combinations is to be treated similarly. If a server supports a single "inherit ACE" flag that applies to both files and directories, receives a request set an NFSv4 ACL with ACE ACE4_FILE_INHERIT_ACE set but ACE4_DIRECTORY_INHERIT_ACE not set, it MUST reject the request with NFS4ERR_INVAL or NFS4ERR_ATTRNOTSUPP.

8.4.1. Details Regarding ACE Flag Bits

Consensus needed (Item #13b),through end of section]: Of the following bits, five are related to inheritance, These are ACE4_FILE_INHERIT_ACE, ACE4_DIRECTORY_INHERIT_ACE, ACE4_NO_PROPAGATE_INHERIT_ACE, ACE4_INHERIT_ONLY_ACE, and ACE4_INHERITED_ACE. Because of the large number of inheritance-related bits, it is important to be aware of:

* What types of objects these bits are used for and how the object type affects their use.

* How these bits interact with one another or might not have any effect if other bits are not set.

* How these bits are copied or modified when inheritable ACEs are transferred as part of server-effected ACE propagation.

* How they affect the existence of inheritance and/or authorization/ monitoring

ACE4_FILE_INHERIT_ACE
: Any non-directory file in any sub-directory will get this ACE inherited.

  Support for this flag bit is OPTIONAL.

  With regard to control of inheritance/authorization:

  * There is no point to specifying this bit for a non-directory object, since there are no objects to inherit ACEs from objects which are not directories.

  * When the flag is set in a directory and a file is created within the directory, the ACE is inherited by the new object, with no change to the flags except the possible resetting of this flag for non-directories and the setting of ACE4_INHERITED_ACE.

* When the "who" value is CREATOR_OWNER@, the inherited "who" value is replaced by the actual owner.

ACE4_DIRECTORY_INHERIT_ACE

Can be placed on a directory and indicates that this ACE is to be added to each new sub-directory created.

Support for this flag bit is OPTIONAL.

If this flag is set in an ACE in an NFSv4 ACL attribute to be set on a non-directory file system object, the operation attempting to set the ACL to needs to fail as described below.

[Author Aside (Items #4g)]: Since I am unable to guess what might by "valid reasons" to bypass the recommendation in the paragraph, keeping this as a residual "SHOULD" for now. This could change if it is determined that the recommendation was never bypassed in existing server implementations, or if a specific valid reason to bypass the recommendation is found.

With regard to control of inheritance/authorization:

* As with ACE4_FILE_INHERIT_ACE, There is no point to specifying this bit for a non-directory object, since there are no directories to inherit ACEs from object which are not directories.

* This flag and ACE4_FILE_INHERIT_ACE can be set and reset independently.

ACE4_NO_PROPAGATE_INHERIT_ACE

Can be placed on a directory. This flag tells the server that inheritance of this ACE is to stop at newly created child directories.

To effect this semantics, the flag ACE4_DIRECTORY_INHERIT_ACE is reset in inherited ACEs. There is no use for ACE4_NO_PROPAGATE_INHERIT_ACE so it is likely to be reset as well.

While support for this flag bit is formally OPTIONAL, it has no use if ACE4_DIRECTORY_INHERIT_ACE is not supported and is REQUIRED if it is.

There is no point in setting this flag if ACE4_DIRECTORY_INHERIT_ACE is not set. As discussed below, it is not certain whether servers are to ignore this or reject ACEs in which this combination appears.

ACE4_INHERIT_ONLY_ACE
Can be placed on a directory but indicates that ACEs with the flags set do not affect authorization or monitoring.

Such ACE sonly affect inheritance of newly created objects (server-effected propagation) or the later propagation of inheritable ACEs as a result of ACE or naming tree changes. ALLOW and DENY ACEs with this bit set do not affect access to the directory, and AUDIT and ALARM ACEs with this bit set do not trigger log or monitoring events. Such ACEs only take effect once they are applied (with this bit cleared) to newly created files and directories as specified by the ACE4_FILE_INHERIT_ACE and ACE4_DIRECTORY_INHERIT_ACE flags.

If this flag is present on an ACE, but neither ACE4_DIRECTORY_INHERIT_ACE nor ACE4_FILE_INHERIT_ACE is present, then an operation attempting to st the ACE needs to fail as described below.

ACE4_SUCCESSFUL_ACCESS_ACE_FLAG and ACE4_FAILED_ACCESS_ACE_FLAG
The ACE4_SUCCESSFUL_ACCESS_ACE_FLAG (SUCCESS) and ACE4_FAILED_ACCESS_ACE_FLAG (FAILED) flag bits may be set only on ACE4_SYSTEM_AUDIT_ACE_TYPE (AUDIT) and ACE4_SYSTEM_ALARM_ACE_TYPE (ALARM) ACE types. If during the processing of the file’s NFSv4 ACL, the server encounters an AUDIT or ALARM ACE that matches the principal attempting the OPEN, the server notes that fact, and the presence, if any, of the SUCCESS and FAILED flags encountered in the AUDIT or ALARM ACE. Once the server completes the ACL processing, it then notes if the operation succeeded or failed. If the operation succeeded, and if the SUCCESS flag was set for a matching AUDIT or ALARM ACE, then the appropriate AUDIT or ALARM event occurs. If the operation failed, and if the FAILED flag was set for the matching AUDIT or ALARM ACE, then the appropriate AUDIT or ALARM event occurs. Either or both of the SUCCESS or FAILED can be set, but if neither is set, the AUDIT or ALARM ACE is not useful.

The previously described processing applies to ACCESS operations even when they return NFS4_OK. For the purposes of AUDIT and ALARM, we consider an ACCESS operation to be a "failure" if it fails to return a bit that was requested and supported.

ACE4_IDENTIFIER_GROUP
Indicates that the "who" refers to a GROUP as defined under UNIX or a GROUP ACCOUNT as defined under Windows. Clients and servers MUST ignore the ACE4_IDENTIFIER_GROUP flag on ACEs with a who value equal to one of the special identifiers outlined in Section 8.4.3.

Support for this flag bit is REQUIRED.

ACE4_INHERITED_ACE
[Consensus needed (Items #123d, #125a), of definition]:

This bit is often set when ACE's are propagated as part of server-managed inheritance. In addition, there are occasions in which this bit is set in ACEs being set as part of client-managed (often called "automatic") inheritance.

When set, indicates that this ACE is either:

* inherited from a parent directory, as part of inheriting an ACE marked ACE4_FiLE_INHERIT_ACE or ACE4_DIRECTORY_INHERIT_ACE.

  This ACE is created as part of server-managed ACE inheritance.

  While it might seem that this contradicts the inherit-only character of such inherited ACEs, since such ACEs have effects that direct authorization rather than inheritance only. Nevertheless, this combination is used so the "only" cannot be taken literally.

* Has received an ACE with this flag bit set in a sacl or dacl attribute.

  This ACE is created as part of client-managed (sometimes referred to as "automatic") ACE inheritance.

  In this case, the client will typically create inherited ACEs to be used for authorization and monitoring to reflect the inheritable ACEs in the encompassing directories irrespective of the relative timing of the creation of the inheritable ACE and the creation of the subsidiary object.

A server that supports ACE4_INHERIT_ONLY will place this flag on any ACEs inherited from the parent directory when creating a new object. Client applications will use this in performing client-managed inheritance. Clients and servers MUST clear this bit in the acl attribute; it may only be used by the server in the dacl and sacl attributes.

The use of this flag in inherited ACEs that do not use ACE4_INHERIT_ONLY is not required since the client can assume such ACE4 are inherited. Server MAY set such flags on such inherited ACEs.

When ACEs have either ACE4_FILE_INHERIT_ACE or ACE4_DIRECTORY_INHERIT_ACE set, the ACE affects inhritance. If ACE4_INHERIT_ONLY is not set, the ACE controls both hinheritance and authorization. Normally, ACEs inherited by subordinate objects receive an ACE which controls both functions as well. However if the "who" value is CREATOR_OWNER@ or CREATOR_GROUP@, there is a need to split these funcions into two ACEs, withone cotrolling inheritance retaining the orginsl "who" value an the onr controllinguthorization/ monitoring directed at the specfic user/group whic created the object.

There are a number of situations listed below in which certain flags or combinations are essentially meaningless. Although, in some such cases, previous specifications have stated that NFS4ERR_ATTRNOTSUPP "SHOULD" be returned, that appoach cannot continue for the following reasons:

* The use of "SHOULD"is essentially meaningless since it is not made clear what might be "valid" reasons to bypass the recommendation or what the consequences that need to be understood when it is bypassed.

* The fact that this is not a valid use of NFS4ERR_ATTRNOTSUPP.

* The fact that a small set of invalid/useless combinations are dealt with, leading to the unstated assumption that other invalid/ useless combinations are to be either ignored or modified to create a valid combination.

In the following cases, "SHOULD" be returned in the cases listed below. In this context, the only valid reason to bypass the recommendation is possible reliance on earlier standards-track specfication that eithr stated something else to be done or that ignored the case, giving rise to the assumption that there was no need to treat the situation as an error.

* Setting of any of the bits ACE4_FILE_INHERIT_ACE, ACE4_DIRECTORTY_INHERIT_ACE, ACE4_NO_PROPAGATE_INHERIT_ACE, or ACE4_INHERIT_ONLY in an ACE associated with a non-directory file object.

* Setting of ACE4_INHERIT_ONLY in an ACE in which neither ACE4_FILE_INHERIT_ACE nor ACE4_DIRECTORTY_INHERIT_ACE is set.

* Setting ACE4_NO_PROPAGATE_INHERIT_ACE in an ACE in which ACE4_DIRECTORTY_INHERIT_ACE is not set.

8.4.2. ACE Flag Support Discovery

[Consensus Needed (Items #13c, #105r), Entire Section]:

The ability to determine what ACE flags are supported depends on information reported by the ACL_Choice attribute.

Because support for all of these flags is OPTIONAL, clients interacting with servers on which the ACL_Support attribute is not supported have no access to extensions, such as those involving ACL inheritance, that require ACE flag support. This problem is exacerbated because there is no way to test for support by using lags and depending on the existence of an error return to resolve the question.

When the ACL_Choice attribute is supported, support for ACE flags can be determined as described below:

* When the flag bit ACC4BN_INHFULL is set, it can be assumed that support for the ACE flags ACE4_FILE_INHERIT_ACE, ACE4_DIRECTORY_INHERIT_ACE, ACE4_INHERIT_ONLY_ACE, AND ACE4_NO_PROPAGATE_INHERIT_ACE is present.

* When the flag bit ACC4BN_IN1BIT is set, it can be assumed that support for the ACE flags ACE4_FILE_INHERIT_ACE, ACE4_DIRECTORY_INHERIT_ACE, ACE4_INHERIT_ONLY_ACE, AND ACE4_NO_PROPAGATE_INHERIT_ACE although there is no support for dealing separately with inheritance in newly created subdirectories and other objects.

  In this case the server does not support ACEs in which the flags ACE4_FILE_INHERIT_ACE and ACE4_DIRECTORY_INHERIT_ACE have different settings. While such combinations are normally rejected, the server is allowed to modify the ACEs to make these settings the same if ACC4BN_RVINV is set. In addition, such ACEs can be accepted but not enforced based on the settings of the data item ACC4IN_STOREUA.

8.4.2.1. ACE Flag Extension

[Consensus Needed (Item #13d), Entire Section]:

Although, in general, previously unused bits within flags words such as the flag field within an ACE can easily be added in extensions, the way that the flag field within is defined creates unusual difficulties, since we cannot be sure that undefined bits in these words are always zero. As a consequence, the definition of additional flags must satisfy one the following constraints:

* Use of the extension needs to be conditioned on support for the Aclfeature attribute and the server setting the flag AF4FLAG_UNDEF0 in the af4flags word returned.

* Definition of the extension has to occur in a future minor version that requires that unused bits in an ACE flag word always be sent not set.

Assuming the flag can be defined as discussed above are satisfactorily dealt with, definitions of new flags needs the following items to be specified:

* An XDR definition of the value of the flag assigning an used flag value to a symbol typically beginning with "ACE4_".

* A definition of what the flag indicates.

* Discussion of the pattern of use of the flag, including whether it is set by the client and used by the server, the reverse, or both.

* For flags set by the client and interpreted by the server, specification of a means by which the client can determine whether the flag is supported.

  This can take the form of a new bit with that purpose within the af4flags word, the support for or value returned by an OPTIONAL attribute, or other means that the client can easily determine whether support is present.

8.4.3. ACE Who

[Consensus Needed (Items #123e, #124d), For entire section]:

The "who" field of an ACE is an identifier that specifies the principal or principals to whom the ACE applies. It may refer to a user or a group, with the flag bit ACE4_IDENTIFIER_GROUP specifying which of the two is referred to.

There are several special identifiers that need to be understood universally, rather than in the context of a particular DNS domain.

[Author Aside, including list]: The above paragraph is OK, but the following issues regarding these special identifiers need to be addressed:

* Lack of clarity about the question of which of these special identifiers have to be supported and for which support is OPTIONAL.

* Lack of clarity about which special identifiers can be understood by NFSv4.

* Confusion of "authentication" and "identification".

[Previous treatment (Item #50b)]: Some of these identifiers cannot be understood when an NFS client accesses the server, but have meaning when a local process accesses the file. The ability to display and modify these permissions is permitted over NFS, even if none of the access methods on the server understands the identifiers.

[Consensus Needed (Item #50b)]: Server support for the special identifiers "OWNER", "GROUP", and "EVERYONE" is REQUIRED. For others support is OPTIONAL with information regarding support discovery appearing in Section 8.4.3.1

[Consensus Needed (Item #50b)]: Some of these identifiers, such as "NETWORK", "DIALUP", "INTERACTIVE", "BATCH", and "SERVICE" cannot be reliably understood when an NFS client accesses the server, but might have meaning when a local process accesses the file or when protocols other than NFSv4 are used As a result, when ACEs containing these who values are encountered, a server that supports these values when NFSv4 request are processed is free to make its own judgment as to whether any particular request will be treated as matching.

[Consensus Needed (Item #50b)]: The ability to display and modify these OPTIONAL principal references is provided for by NFSv4, even though they are not, in many cases, useful when processing NFSv4 requests,

Who	Description
OWNER	The owner of the file.
GROUP	The group associated with the file. Note that the mode bits for the group do not apply to the owner of the file while the owner of the file is a member of GROUP@
EVERYONE	[Previous treatment (Item #50b)]: The world, including the owner and owning group. [Consensus Needed (Item #50b)]: All requesters, including the owner, members of the owning group, and requests for which no user information is available.

CREATOR_OWNER	ACEs with this special "who" value can only occur in inherit-only ACEs. When they appear in other ACEs, the ACL is rejected with error NFS4ERR_INVAL. The addition of such an ACE, because it is inherit-only, has no effect on the authorization or related security functions associated with the target directory. However, when file and subdirectories are created within this directory, the handling of inherited ACLs is affected to allow, deny, audit or trigger alarms for the principal that becomes the owner at the creation of the object. Unlike, the case of the value OWNNER@, this principal remains the same even if the owner is changed. Nevertheless, when objects are created within a sub-directory, the original ACE with "who" value CREATOR_OWNER@ is inherited.
CREATOR_GROUP	ACEs with this special "who" value can only occur in inherit-only ACEs. When they appear in other ACEs, the ACL is rejected with error NFS4ERR_INVAL. The addition of such an ACE, because it is inherit-only, has no effect on the authorization or related security functions associated with the target directory. However, when files and subdirectories are created within this directory, the handling of inherited ACLs is affected to allow, deny, audit or trigger alarms for the principals within the group that becomes the owning group at the creation of the object. Unlike, the case of the value GROUP@, this group remains the same even if the owning group is changed. Nevertheless, when objects are created within a sub-directory, the original ACE with "who" value CREATOR_GROUP@ is inherited.
OWNER_RIGHTS	Acts similarly to OWNER@ with one important exception. That exception concern the handling of permissions that the filesystem grants to a principal by virtue of its role as the owner of the file (e.g. permission to modify the ACL.

	Such default permissions do not normally require explicit mention in an ACE. When the "who" value OWNER_RIGHTS@ is used, such default permissions are cancelled and, if not explicitly granted are not available to the owner. When the ACL_Choice attribute is supported and the ACC4IN_INFMB data item is present then these rights are determined using acmi_defown field modified to remove mask bits that are in neither the acms_valid nor the acms_separate fields for the object’s type. When the ACL_Choice attribute is not supported or the ACC4IN_INFMB data item is not present then, the client can assume that ACE4_READ_ACl | ACE$_WRITE_ACL is to be used as the set of user-based default rights.
INTERACTIVE	Accessed from an interactive terminal.
NETWORK	Accessed via the network.
DIALUP	Accessed as a dialup user to the server.
BATCH	Accessed from a batch job.
ANONYMOUS	[Author Aside]: Although AUTH_NONE requests, are included here it is not clear whether AUTH_SYS requests, particularly those issued without client peer authentication should be included. Although I feel that they are not truly authenticated, it is hard to be sure of the intention given the way term "authenticated" has been used in earlier specifications. Also hard to resolve is the status of "nobody@domain" which is intended to be anonymous but may well be authenticated, as some other user. [Author Aside]: For now, am doing the best I can, given that no implementations of these have been found. When they are, will need to consider revision. If none are found, could consider deleting these. In any case, think we will wind up treating these as opposites, unless forced to do otherwise by existing implementations.

	[Consensus Needed (Item #50b)]: Accessed without any authentication of the user principal (e.g., via AUTH_NONE). Also can include user defined as anonymous such as those which result from root-squashing, regardless of the quality of authentication
AUTHENTICATED	[Author Aside]: As for the previous case, am doing the best I can given that no implementation of these have been found. [Consensus Needed (Item #50b)]: Any authenticated user except those to be treated as anonymous (opposite of ANONYMOUS).
SERVICE	Accessed from a system service.

Table 5

To avoid conflict, these special identifiers are distinguished by an appended "@" and will appear in the form "xxxx@" (with no domain name after the "@"), for example, ANONYMOUS@.

{Previous treatment (Item #51a)]: The ACE4_IDENTIFIER_GROUP flag MUST be ignored on entries with these special identifiers. When encoding entries with these special identifiers, the ACE4_IDENTIFIER_GROUP flag SHOULD be set to zero.

[Author Aside]: I don’t understand what might be valid reasons to ignore this or how a server would respond in the case the that it was ignored.

[Consensus Needed (Item #51a)]: The ACE4_IDENTIFIER_GROUP flag MUST be ignored on entries with these special identifiers. When encoding entries with these special identifiers, the ACE4_IDENTIFIER_GROUP flag should be set to zero.

It is important to note that "EVERYONE@" is not equivalent to the UNIX "other" entity. This is the case because, by definition, UNIX "other" does not include the owner or owning group of a file. "EVERYONE@" means literally everyone, including the owner or owning group.

8.4.3.1. ACE Who Value Support Discovery

[Consensus Needed (Items #50c, #105s, #117j, #123f, #124e), For entire section]:

[Author Aside (Item #50c), to the end of the bulleted list]: Given that we have not yet encountered any implementations of these special OPTIONAL values. When we complete our analysis of existing implementations, we might update this in various ways:

* If it is the case that one or more of these special who values is never accepted, the group will need to consider whether it make sense to delete it now.

  While there is no time gap that, by itself, would justify unimplemented feature, the working group could reasonably conclude that the absence of implementation for multiple decades could consider that there is no real need for he feature.

* If an entire class of special who values (e.g. auth-related or miscellaneous) were never implemented, it would be possible streamline this section to delete it together with the flag bits in af4flags supporting its discovery

The support for special who values is indicated by the following flags when ACL_Choice is supported, but when that attribute is not supported there is no way for the client to determine whether support is present.

* The flag bit ACC4BN_AUTHWHO indicates the presence of support for the special who values INTERACTIVE@, NETWORK@, BATCH@, DIALUP@, and SERVICE@.

* The flag bit ACC4BN_OTHWHO indicates the presence of support for the special who values ANONYMOUS@ and AUTHENTICATED@.

* The flag bit ACC4BN_CRWHO indicates the presence of support for the special who values CREATOR_OWNER@ and CREATOR_GEOUP@,

* The flag bit ACC4BN_ORIGHTSWHO indicates the presence o of support for the special who value OWNER_RIGHTS@.

8.4.3.2. ACE Who Value Extension

[Consensus Needed (Item #50d), for entire section]:

Because use of unknown who values is defined as returning NFS4ERR_BADOWNER, standard track documents defining extensions to extensible minor versions can define new special who values. Definitions of such new values need to include the following:

* The string(s) to serve with each together with an appended "@" as the new special who value. The specification should be in term of the Unicode characters. If it is desired that case-insensitive or normalization-form-insensitive string matching is desired, then multiple strings should be specified rather than specifying the type of code- insensitivity desired.

* A description of how it is to be determined whether a given NFSv4 request matches the new special who value. In this context, "never", indicating such ACEs are to be ignored, is acceptable.

  How these affect handling of non-NFSv4 requests can be treated as out of scope.

* A description of how support for this new value is to be ascertained. This can take the form of the specification of the new value as auth-related, miscellaneous, or the identification of a new bit within af4whoinf values that indicates whether support is present.

9. Processing Access Control Entries

To determine if a request succeeds, the server processes each nfsace4 entry of type ALLOW or DENY in turn as ordered in the array. Only ACEs that have a "who" that matches the requester are considered. An ACE is considered to match a given requester if at least one of the following is true:

* The "who’ designates a specific user which is the user making the request.

* The "who" specifies "OWNER@" and the user making the request is the owner of the file.

* The "who" designates a specific group and the user making the request is a member of that group.

* The "who" specifies "GROUP@" and the user making the request is a member of the group owning the file.

* The "who" specifies "EVERYONE@".

* The "who" specifies "INTERACTIVE@", "NETWORK@", "DIALUP@", "BATCH@", or "SERVICE@" and the requester, in the judgment of the server, feels that designation appropriately describes the requester.

* The "who" specifies "ANONYMOUS@" or "AUTHENTICATED@" and the requestor's authentication status matches the who, using the definitions in Section 8.4.3

Each ACE is processed until all of the bits of the requester's access have been ALLOWED. Under most circumstances, once a bit (see below) has been ALLOWED by an ACCESS_ALLOWED_ACE, it is no longer considered in the processing of later ACEs. However if the server chooses not to allow partial satisfaction of ALLOW ACEs (see below), this does not happen and the operation can only be allowed of a single ACE allows all required actions. If an ACCESS_DENIED_ACE is encountered where the requester's access still has bits not ALLOWED in common with the "access_mask" of the ACE, the request is denied. When the ACL is fully processed, if there are bits in the requester's mask that have not been ALLOWED or DENIED, access is denied.

Unlike the ALLOW and DENY ACE types, the ALARM and AUDIT ACE types do not affect a requester's access, and instead are for triggering events as a result of a requester's access attempt. AUDIT and ALARM ACEs are processed only after processing ALLOW and DENY ACEs if any exist. This is necessary since the handling of AUDIT and ALARM ACEs are affected by whether the access attempt is successful.

[Previous Treatment]: The NFSv4.1 ACL model is quite rich. Some server platforms may provide access-control functionality that goes beyond the UNIX-style mode attribute, but that is not as rich as the NFS ACL model. So that users can take advantage of this more limited functionality, the server may support the acl attributes by mapping between its ACL model and the NFSv4.1 ACL model. Servers must ensure that the ACL they actually store or enforce is at least as strict as the NFSv4 ACL that was set. It is tempting to accomplish this by rejecting any ACL that falls outside the small set that can be represented accurately. However, such an approach can render ACLs unusable without special client-side knowledge of the server's mapping, which defeats the purpose of having a common NFSv4 ACL protocol. Therefore, servers should accept every ACL that they can without compromising security. To help accomplish this, servers may make a special exception, in the case of unsupported permission bits, to the rule that bits not ALLOWED or DENIED by an ACL must be denied. For example, a UNIX-style server might choose to silently allow read attribute permissions even though an ACL does not explicitly allow those permissions. (An ACL that explicitly denies permission to read attributes should still be rejected.)

[Author Aside]: While the NFSv4.1 provides features that many might not need or use, it is the one that the working group adopted by the working group, and I have to assume that alternatives, such as the withdrawn POSIX ACL proposal were considered but not adopted. The

phrase "unsupported permission bits" with no definition of the bit whose support might be dispensed with, implies that the server is free to support whatever subset of these bits it chooses. As a result, clients would not be able to rely on a functioning server implementation of this OPTIONAL attribute. If there are specific compatibility issues that make it necessary to allow non-support of specific mask bits, then these need to be limited and the client needs guidance about determining the set of unsupported mask bits.

[Previous Treatment]: The situation is complicated by the fact that a server may have multiple modules that enforce ACLs. For example, the enforcement for NFSv4.1 access may be different from, but not weaker than, the enforcement for local access, and both may be different from the enforcement for access through other protocols such as SMB (Server Message Block). So it may be useful for a server to accept an ACL even if not all of its modules are able to support it.

[Author Aside]: The following paragraph does not provide helpful guidance and takes no account of the need of the client to be able to rely on the server implementing protocol-specifying semantics and giving notice in those cases in which it is unable to so

[Previous Treatment]: The guiding principle with regard to NFSv4 access is that the server must not accept ACLs that appear to make access to the file more restrictive than it really is.

10. Combining Authorization Approaches

10.1. Background for Combining of Authorization Approaches

Both [RFC7530] and [RFC8881] contain material relating to the need, when both mode and ACL attributes are supported, to make sure that the values are appropriately coordinated. Despite the fact that these discussions are different, they are compatible and differ in only a small number of areas relating to the existence or absence of the set-mode-masked attribute.

Such coordination is necessary is necessary since it is expected that servers providing both sets of attributes will encounter users who have no or very limited knowledge of one and need to work effectively when other users changes that attribute. As a result, these attributes cannot each be applied independently since that would create an untenable situation in which some users who have the right to control file access would find themselves unable to do so.

This section and included subsections discuss issues related to the use of the NFSv4 ACL model. Similar issues, as they apply to the use of Access Control Lists defined as part of later extensions are to be dealt with by the documents defining those extensions.

[Author Aside]: From this point on, all paragraphs in this section, not other annotated are to be considered part of Consensus Item #63a. The description in this section of changes to be made reflects the author's view of the many aspects of this issue and related issues. The eventual approach chosen might have to be adjusted based on working group decisions.

As a result, in this document, we will have a single treatment of this issue, in Sections 10.3 through 10.11. In addition, an NFSv4.2-based extension related to attribute co-ordination will be described in Section 10.12.

How this treatment is to be used depends on the needs of the reader:

* This treatment is appropriate for those concerned with the approach to take to these attribute coordinations going forward.

* Those concerned with existing implementations and understanding the changes of the handling of these matters compared to that in previous minor version specifications need to be aware of other material.

  The issues that prompted a significantly different approach to the specification of these matters are explained in Section 10.2.

  The discussion of the computation of a mode from an ACL, discussed in Section 10.5 would need supplementation in Appendices B.2 and B.3.

10.2. Problems in Previous Handling of the Combining of Authorization Approaches

The current NFSv4.0 and NFSv4.1 descriptions of this co-ordination share an unfortunate characteristic in that they are both written to give server implementations a broad latitude in implementation choices while neglecting entirely the need for clients and users to have a reliable description of what servers are to do in this area.

As a result, one of the goals of this new combined treatment will be to limit the uncertainty that the previous approach created for clients, while still taking proper account of the possibility of compatibility issues that a more tightly drawn specification might give rise to.

The various ways in which these kinds of issues have been dealt with are listed below together with a description of the needed changes proposed to address each issue.

* In some cases, the term "MAY" is used in contexts where it is inappropriate, since the allowed variation has the potential to cause harm in that it leaves the client unsure exactly what security-related action will be performed by the server.

  The new treatment will limit use of MAY to cases in which it is truly necessary, in order to give clients proper notice of cases in which server behavior cannot be determined and limit the work necessary to deal with a large array of possible behaviors.

* There are also cases in which no RFC2119-defined keywords are used but it is stated that certain server implementations do a particular thing, leaving the impression that that action is to be allowed, just as if "MAY" had been used.

  If the flexibility is necessary, MAY will be used. In other cases, SHOULD will be used with the understanding that maintaining compatibility with clients that have adapted to a particular approach to this issue is a valid reason to bypass the recommendation. However, in no case will it be implied, as it is in the current specifications, that the server MAY do whatever it chooses, with the client having no option but to adapt to that choice.

* There was a case, in description of the computation of mode values corresponding to ACLs, in which the term "SHOULD" was explicitly used intentionally, without it being made clear what the valid reasons to ignore the guidance might be, although there was a reference to servers built to support the now-withdrawn draft definition of POSIX ACLs, which are part of what is referred to in this document as "UNIX ACLs", as described in Section 4.1 of [I-D.dnoveck-nfsv4-security]. A discussion of the issues for support of for these ACLs appears in Section 6.1.

[Author Aside]: Despite the statement, now cited in Section 10.3, that this was to accommodate implementations "POSIX" ACLs, it now appears that this was not complete. I've been given to understand that this was the result of two groups disagreeing on the appropriate mapping from ACLs, and specifying both, using the "intentional" "SHOULD" essentially as a MAY, with the text now in Section 10.3 discouraging such use as potentially confusing, not intended to be taken seriously. Since the above information might not be appropriate in a standards-track RFC, we intend to retain this as an Author Aside which the working group might consider as it discusses how to navigate our way out of this situation.

The new approach will use the term "RECOMMENDED" without use of the confusing term "intentional". The valid reasons to bypass the recommendation will be clearly explained as will be the consequences of choosing to do other than what is recommended.

* There are many case in which the terms "SHOULD" and "SHOULD NOT" are used without any clear indication why they were used. In this situation it is possible that the "SHOULD" was essentially treated as a "MAY" but also possible that servers chose to follow the recommendation.

  In order to deal with the many uses of these terms in Section 10 and included subsections, which have no clear motivation, it is to be assumed that the valid reasons to act contrary to the recommendation given are the difficulty of changing implementations based on previous analogous guidance, which may have given the impression that the server was free to ignore the guidance for any reason the implementer chose. This allows the possibility of more individualized treatment of these instances once compatibility issues have been adequately discussed.

  [Author Aside]: In each subsection in which the interpretation of these terms in the previous paragraph applies there will be an explicit reference to Consensus Item #63, to draw attention to this change, even in the absence of modified text.

10.3. Needed Attribute Coordination

On servers that support the acl or dacl attributes, together with the mode attribute, the server needs to keep the two sets of attributes consistent with one another. This is because a major element of each controls file access authorization so that, when one of these is set, the other cannot be inconsistent. In the case of NFSv4, when one is set, the other needs to be adjusted to avoid inconsistency, taking appropriate note of the fact that each attribute has data that control functions beyond access authorization for the associated

object.

* The mode attribute, in addition to the nine bits devoted to file access authorization contains a set of bits reserved for client use as described in Section 5.3.1 of [I-D.dnoveck-nfsv4-security].

* When the ACL-related attributes are supported, they control file access authorization but also other matters that are separate from matters controlled by the low-order bits of the mode attribute. In the cases of the acl and sacl attributes, these include the functions discussed in Section 11, while for the acl, dacl, and sacl attributes, there are matters connected with ACL inheritance.

Also important is the fact that each attribute group uses a different approaches to authorization granularity.

* The permission granularity provided by the ACE mask word is considerably finer than that provided by the three permission bits derived from POSIX.

  Complicating matters, the granularity provided by particular servers can be different, ranging from the coarse-grained approach provided by POSIX to the fine-grained model in effect when all of the defined ACE mask bits control the permission to effect distinct sets of target actions.

  There are a number of mask bits for which the set of covered actions has not been sufficiently clarified to be a basis for interoperable implementations.

* The principal sets to which permissions might be granted are considerably more restricted in the POSIX-based model.

  In the POSIX-based model, there are three sets of principals that can have different permission sets.

Because of these differences in functionality, there are corresponding issue that are important to note when these distinct attributes are to be kept consistent

* Each mode has a corresponding ACL that provides the same permissions.

  In cases in which the owning group has fewer permission than the owner or others have fewer permissions than the owning group (aka "reverse-slope-mode), this ACL needs to contain DENY ACEs,

* There are ACLs that cannot be represented by a corresponding mode value.

As a result, the following coordinating actions are needed:

* When the mode attribute is modified,

  See Sections 10.7 through 10.9 for detailed discussion of these matters.

10.4. Adapting to the Previous Approach to Attribute Coordination

As discussed in more detail in Section 10.3, servers that support acl or dacl attributes, together with the mode attribute, the server needs to keep the two attributes consistent with one another.

[Previous Treatment (Item #63b)]: When a mode attribute is set on an object, the ACL attributes may need to be modified in order to not conflict with the new mode. In such cases, it is desirable that the ACL keep as much information as possible. This includes information about inheritance, AUDIT and ALARM ACEs, and permissions granted and denied that do not conflict with the new mode.

[Author Aside]: one the things that this formulation leaves uncertain, is whether, if the ACL specifies permission for a named user group or group, it "conflicts" with the mode. Ordinarily, one might think it does not, unless the specified user is the owner of the file or a member of the owning group, or the specified group is the owning group. However, while some parts of the existing treatment seem to agree with this, other parts, while unclear, seem to suggest otherwise, while the treatment in Appendix B.4 is directly in conflict.

[Previous Treatment (Item #26a)]: The server that supports both mode and ACL must take care to synchronize the MODE4_*USR, MODE4_*GRP, and MODE4_*OTH bits with the ACEs that have respective who fields of "OWNER@", "GROUP@", and "EVERYONE@".

[Author Aside]: This sentence ignores named owners and group, giving the impressions that there is no need to change them.

[Previous Treatment (Item #26a)]: This way, the client can see if semantically equivalent access permissions exist whether the client asks for the owner, owner_group, and mode attributes or for just the ACL.

[Author Aside, Including List:] The above sentence, while hard to interpret for a number a reasons, is worth looking at in detail because it might suggest an approach different from the one in the previous sentence from the initial paragraph for The Previous Treatment of Item #26a.

* The introductory phrase "this way" adds confusion because it suggests that there are other valid ways of doing this, while not giving any hint about what these might be.

* It is hard to understand the intention of "client can see if semantically equivalent access permissions" especially as the client is told elsewhere that he is not to interpret the ACL himself.

* If this sentence is to have any effect at all it, it would be to suggest that the result be the same "whether the client asks for the owner, owner_group, and mode attributes or for just the ACL."

  If these are to be semantically equivalent it would be necessary to delete ACEs for named users, which requires a different approach from the first sentence of the original paragraph.

[Previous Treatment (Item #27a)]: Much depends on the method in specified Appendix B.2. Many requirements refer to this section. It needs to be noted that the methods have behaviors specified with "SHOULD" and that alternate approaches are discussed in Appendix B.3. This is intentional, to avoid invalidating existing implementations that compute the mode according to the withdrawn POSIX ACL draft (1003.1e draft 17), rather than by actual permissions on owner, group, and other.

[Consensus (Item #27a)]: In performing the coordination discussed in this section, the method used to compute the mode from the ACL has an important role. In this regard, the method described Section 10.5 is to be used. This method allows considerable variation including that motivated by a decision allow computation using the method mandated by withdrawn POSIX ACL draft (1003.1e draft 17). Since this means that a client, having no way of determining the method the server uses may face interoperability difficulties in moving between servers which approach this matter differently, these problems need to be accepted unless and until the working group revises the protocol to require more uniform treatment. In any case, the ACL_Choice attribute allows the client to determine which approach is used by the current server. A more complete discussion of handling of the UNIX ACLs in general is to be found in Section 6.1.

10.5. Computing a Mode Attribute from an ACL (proposed)

[Consensus Needed (Items #27b, #28b, #61f, #105t, #110f, #126f), Through end of section]:

The following method (or another one providing exactly the same results) SHOULD be used to calculate the MODE4_R*, MODE4_W*, and MODE4_X* bits of a mode attribute. In this case, the only valid reason to bypass the recommendation is implementor reliance on previous specifications which left this to implementor choice or ignored the cases of the owner having less access than the owning group or the owning group having less access than others. Further, in implementing or the maintaining an implementation previously believed to be valid, the implementor needs to be aware that this will result in invalid values in some uncommon cases.

First, for each of the sets of mode bits (i.e., user, group other, process the ACL in order, with the result in each case with a specific evaluation procedure depending on the specific set of mode bits being determined. The result will be two acemask4 words which are referred to below as "Allowed" and "Denied".

For each set there will be one or more special identifiers considered in a positive sense so that ALLOW and DENY ACE’s are considered in arriving at the mode bit. In addition, for some sets of bits, there will be one or more special identifiers to be considered only in a negative sense, so that only DENY ACE’s are considered in arriving at the mode it.

The users to be considered are as follows:

* For the owner bits, "OWNER@", @GROUP, and "EVERYONE@" are to be considered, all in a positive sense.

* For the group bits, "GROUP@" and "EVERYONE@" are to be considered, both in a positive sense, while "OWNER@" is to be considered in a negative sense.

  In some cases, ALLOW ACEs specifying named users and groups are considered, in a positive sense. Fundamentally the decision whether to do so or not is up to the server, although it needs to always make the same choice for all object in a given file system.

* For the bits for others, "EVERYONE@" is to be considered in a positive sense, while "OWNER@" and "GROUP@" are to be considered in a negative sense.

Regarding the phrases "considered in a ...sense", the following actions are indicated:

* When the ACE to be considered in a positive sense is an ALLOW ACE, the associated mask m is used in AddToAllowed(m), as described below. When it is a DENY ACE, the associated mask m is used in AddToDenied(m), as described below.

* When the ACE to be considered in a negative sense is an ALLOW ACE, the associated mask ignored When it is a DENY ACE, the associated mask m is used in AddToDenied(m), as described below.

```
<CODE BEGINS>
void AddToAllowed(acemask4 m)
{
    acemask4 ov = Denied & m;

    m &= ~ov;
    Allowed |= m;
};

void AddToDenied(acemask4 m)
{
   acemask4 ov = Allowed & m;
   acemask4 nov = m & ~ov;

   Allowed &= ~ov;
   Denied |= nov;
};

acemask4 GetAllowed(void)
{
   return Allowed & ~Denied;
}
<CODE ENDS>
```

Once these ACL masks are constructed, the mode bits for, user, group, and others can each be obtained as described below:

* The mask to use to determine the privilege bits is obtained using GetAllowed().

* For each of the mode-computation masks, which can be obtained using the BitsNeeded[] element for each mask bit, the associated privilege bit is set if all of the mask bits in the associated mask are allowed.

10.6. Setting Multiple ACL Attributes

In the case where a server supports the sacl or dacl attribute, in addition to the acl attribute, the server MUST fail a request to set the acl attribute simultaneously with a dacl or sacl attribute. The error to be given is NFS4ERR_ATTRNOTSUPP.

10.7. Setting Mode and not ACL

10.7.1. Overview of Setting Mode and not ACL

[Author Aside]: This proposed section is part of Consensus Item #30c and all unannotated paragraphs within it are to be considered part of that Item. Since the proposed text includes support for reverse-slope modes, treats all minor versions together and assumes decisions about handling of ACEs for named users and groups, the relevance of consensus items #26b, #28, and #29 needs to be noted.

[Author Aside]: As with all such Consensus Items, it is expected that the eventual text in a published RFC might be substantially different based on working group discussion of client and server needs and possible compatibility issues. In this particular case, that divergence can be expected to be larger, because the author was forced to guess about compatibility issues and because earlier material, on which it is based left such a wide range of matters to the discretion of server implementers. It is the author’s intention that, as the working group discusses matters, sufficient attention is placed on the need for client-side implementations to have reliable information about expected server-side actions.

[Author Aside, through end of bulleted list]: In this and subsequent sections I have tried to come up with overall recommendations that are as consistent with the previous treatment as I can come up with, as I have done in other areas. In this particular case, I have had more difficulty than in others since this is the existing text treatment is so unclear, making it hard to determine what hard-to-accommodate aspects are intentional. In particular, as the working group discusses this area and accommodate actual implementation, the following difficult issues will need to be focused on:

* The retention or not of ACEs using who values directly designating particular users or groups (as opposed to @OWNER, @GROUP, @EVERYONE). Some of the previous text suggests these are to be retained but I have specified they can be retained only if it is certain that they will have no effect on authorization.

Allowing these to be retained and used would partially erode the control of file authorization by the mode attribute, resulting in a confusing situation.

* The statement in existing specification about avoiding conflicts with the mode derived from the acl is not all that clear about which method is intended. In addition, it is not clear how seriously to take this since the purpose of supporting ACLs is precisely to express authorization patterns that are different from those expressible by all potential mode values and thus inconsistent with all of them.

  [Author Aside (Items #105u, #126g)]: I have allowed multiple modes of computation while providing facilities within ACL_Choice attribute for the client to be informed of server choices in this regard. However, the mapping of privilege bits and ownership to ACE masks derives from the MaskFromPrivVal array as described in Sections 8.3.15 and 8.3.16. This applies even if ACL_Choice is not supported since the server can respond to a mode change based on its own handling of mask bits without the client's participation.

This section describes how ACLs are to be updated in response to actual or potential changes in the mode attribute, when the attributes needed by both of the file access authorization models are supported. It supersedes the discussions of the subject in [RFC7530] and [RFC8881], each of which appeared in Section 6.4.1.1 of the corresponding document.

Despite this supersession, it needs to be understood that previous implementations addressed the issues using relying often on now-superseded statements about the requirements to be satisfied and how these requirements might be met. In light of the existence of these implementations, in defining what would normally be requirements, we use the term "SHOULD" with the understanding that reliance on material in these earlier specifications is a valid reason to bypass the new recommendation.

It is necessary to approach the matter differently than in the past because:

* Organizational changes are necessary to address all minor versions together.

* Those previous discussions are often internally inconsistent leaving it unclear what specification-mandated actions are to be performed.

* In many cases, servers were granted an extraordinary degree of freedom to choose the action to take, either explicitly or via an apparently unmotivated use of "SHOULD", leaving it unclear what might be considered "valid" reasons to bypass the recommendation.

* There appears to have been no concern for the problems that clients and applications might encounter dealing ACLs in such an uncertain environment.

* Cases involving reverse-slope modes were not adequately addressed.

* The security-related effects of SVTX were not addressed.

While that earlier approach might have been considered workable at the time, it made it difficult to devise client-side ACL implementations that incorporated the extensions within NFSv4 ACLs, even if there had been any interest in doing so. In order to enable this situation to eventually be rectified, we will define the preferred implementation here, but in order to provide temporary compatibility with existing implementations based on reasonable interpretations of [RFC7530] [RFC8881]. To enable such compatibility the term "SHOULD" will be used, with the understanding that valid reasons to bypass the recommendation, are limited to implementers' previous reliance on these earlier specifications and the difficulty of changing them now.

When the recommendation is bypassed in this way, it is necessary to understand, that, until the divergence is rectified, or the client is given a way to determine the detail of the server's non-standard behavior, client-side implementations may find it difficult to implement a client-side implementation that correctly interoperates with existing servers that based their implementations on various pieces of the existing text, now superseded.

The fundamental recommendation that needs to be addressed is that when mode bits involved in determining file access authorization are subject to modification, the server MUST, when ACL-related attributes have been set, modify the associated ACEs so as not to conflict with the new value of the mode attribute.

The occasions to which this recommendation applies, vary with the attribute being set and the type of object being dealt with:

* For all minor versions, any change to the mode attribute, triggers this recommendation

* When the set_mode_masked attribute is being set on an object which is not a directory, whether this recommendation is triggered depends on whether any of the nine low-order bits of the mode is included in the mask.

* When the set_mode_masked attribute is being set on a directory, whether this recommendation is triggered depends on whether any of the nine low-order bits of the mode or the SVTX bit is included in the mask of bit whose values are to be set.

[Consensus Needed (Item #105u)]: In order to accommodate servers who based their implementation on the trivial-implementation-remark, servers can bypass the recommendation but need to set the flag ACC4BN_SMJUST3 when the ACL_Choice attribute is supported.

When the recommendation is to be followed, ACLs need to be updated to be consistent with the new mode attribute. When the server does so, it needs to set the flag ACC4BN_SMFULL when the ACL_Choice attribute is supported. The necessary action depends on specific of the ACEs involved.

* In the case of AUDIT and ALARM ACEs, which are used outside of file access authorization, no change needs to be made.

* {Consensus needed (Item #125b), through end of section]: For ALLOW and DENY ACEs, which are marked as for inheritance only, no change needs to be made since such ACEs have no effect on file access authorization for the current file.

  For our purposes, we consider an marked for inheritance only, if it has the flag ACE4_INHERIT_ONLY_ACE set and the flag ACE4_INHERITED_ACE reset

* For ALLOW and DENY ACEs, which are marked as providing ACE inheritance without as for inheritance only, the effect of these ACEs on inheritance needs to be retained while their direct effect on file access authorization needs to be disabled.

  This can be effected by modifying the ACE to be inherit-only or ensuring that such ACEs are never reached when scanning an ACL for file access authorization. For example, a DENY ACE for EVERYONE@ early in the ACL would have this effect

  Because previous specifications did not mention this case, it is possible that implementations exist that do not follow the trivial-implementation-remark but still do not specifically address this case. Such servers need to set the flag ACC4BN_SMOLD when the ACL_Choice attribute is supported.

* The handling of remaining ACEs with a who-value of OWNER@, GROUP@, or EVERYONE@ needs to be adapted to the new mode.

  This could take the form of rewriting them in place or of generating new ACEs at the start of the ACL.

* The effect on file authorization of remaining ACEs whose who-value is a named user needs to be avoided.

  This can be accomplished by rewriting the ACL, eliminating such ACEs or by or ensuring that such ACEs are never reached when scanning an ACL for file access authorization. For example, a DENY ACE for EVERYONE@ early in the ACL would have this effect.

* The effect on file authorization of ACEs whose who-value is one of the other special values defined in Section 8.4.3 are to be left unmodified.

  This can be accomplished by rewriting the ACL, eliminating such ACEs or by or ensuring that such ACEs are never reached when scanning an ACL for file access authorization. For example, a DENY ACE for EVERYONE@ early in the ACL would have this effect.

How these needs are best effected depends on the ACL model implemented. Of particular importance is the existence of DENY ACEs which allow one to force scanning for file access to be stopped at some point while retaining later ACEs to be retained without any possibility that they will affect file access authorization. We discuss three classes of ACL implementations as discussed below.

* For implementation of NFSv4 ACLs and hybrids that contain support for DENY ACEs, implementation suggestions appearing in Section 10.7.3 are provided.

* For implementation of UNIX ACLs, implementation suggestions appearing in Section 10.7.2 are provided.

* For implementation of various hybrid ACLs that do not provide support for DENY ACEs, implementation suggestions appearing in Section 10.7.4 are provided.

All of the abovementioned suggestions share common logic regarding the formation of ACE masks used and how the mode bits are mapped to ACE masks designating allowed actions.

The mask of allowed actions for each of OWNER@, GROUP@, and EVERYONE@ is derived as described below using mode-set mask support information, as described below. For each of the above, the following is done:

* The three privilege bits are extracted for owner, group, and others.

* That set of three bits is combined with a fourth which one in the case of OWNER@ and zero in the other case.

* Those four fits are used to arrive at a shift to get to the appropriate bit in the of mode-set masks.

* All of the mode set mask words are scanned to find those with that specific bit on. For entries in which it is on the associated ACE mask bit is added to the mask being accumulated.

* The resulting mask designate the allowed actions for the specific who being interrogated.

10.7.2. Setting Mode and not ACL in the Unix ACL Case

[Author Aside]: This proposed section is part of Consensus Item #30d and all unannotated paragraphs within it are to be considered part of that Item. Since the proposed text includes support for reverse-slope modes, treats all minor versions together and assumes decisions about handling of ACEs for named users and groups, the relevance of consensus items #26c, #28, and #29 needs to be noted.

When UNIX ACLs are implemented, the absence of support for DENY ACEs forces the ACL to be rewritten in its entirety, rather than have a mode-related section prepended to a mostly unchanged ACL. In addition, the absence of support for DENY ACEs requires special attention to the possible presence of reverse slope mode because OWNER@ is a subset of GROUP@ and both OWNER@ and GROUP@ are subsets of EVERYONE@.

It should be noted however, that the complexity of the rewriting process is reduced because of features not part of the UNIX ACL model:

* The absence of support for ACEs other than ALLOW means only a single ACE type needs to be dealt with.

* The absence of ACE inheritance means that ACEs marked to be inherited or inherit-only, do not exist.

* The absence of support for ACEs with OPTIONAL special who value allow these cases to be ignored as well.

The replacement ACL begins with three ALLOW ACEs for the who values OWNER@, GROUP@, and EVERYONE@. The order in which these are placed in the resultant ACL needs to be adjusted based on the mode value avoid problems with reverse-slope modes. Such problems can arise when a who value processed later contains permission bits not present in previous one so that the later who value, covering a superset of the principals of the earlier one, receives permissions that should not, for example, be granted to the owning user according to the POSIX definition of privileges for the owning group

In order to address this issue the three entries need to be sorted in order of descending privilege, using the inclusion relationship for the privilege bits of each one.

It is possible that two entries will have privilege sets not orderable by inclusion, i.e., neither is a subset of the other. Given the absence of DENY ACEs, the resulting permissions cannot be representing by an ACL, so that the ACL needs to be deleted in this case.

These preliminary ACEs are followed by a series of ACEs derived from the existing ACL with entries copied over or not as described below:

* ACEs with who value of OWNER@, GROUP@, or EVERYONE@ are not copied over.

* ACEs with other who values are copied to the ACL, unmodified.

10.7.3. Setting Mode and not ACL When DENY ACEs are Supported

[Author Aside]: This proposed section is part of Consensus Item #30e and all unannotated paragraphs within it are to be considered part of that Item. Since the proposed text includes support for reverse-slope modes, treats all minor versions together and assumes decisions about handling of ACEs for named users and groups, the relevance of consensus items #26d, #28, and #29 needs to be noted.

This covers in addition to NFSv4 ACLs per se, all cases in which support for DENY ACEs is present. The availability of support for DENY ACEs affects the generation of a new ACL as follows:

* Reverse slope modes do not force a re-ordering of the initial ACEs. To avoid this, each initial ALLOW ACE is paired with a corresponding DENY ACE

* Detailed analysis of the existing ACEs is not necessary since the new ACL will prevent those from ever being referenced in connection with file access authorization. This allows the existing ACL to be appended to the three initial ACE pairs.

The replacement ACL begins with three pairs of ACEs for the who values OWNER@, GROUP@, and EVERYONE@. Each pair consists of an ALLOW ACE for that who value followed by a corresponding DENY ACE with the same who value. The ACE mask for the ALLOW ACE is derived from the corresponding permission bits as described above. The mask for the DENY ACE is the set of mode-related mask bits with the allows mask bits turned off.

These preliminary ACEs are followed by copies of the ACEs within the existing ACL. It is possible, although not necessary, to eliminate, as part of this copy, all ALLOW and DENY ACEs with who values of OWNER@, GROUP@, and EVERYONE@.

10.7.4. Setting Mode and not ACL When DENY ACEs are Not Supported

[Author Aside]: This proposed section is part of Consensus Item #30f and all unannotated paragraphs within it are to be considered part of that Item. Since the proposed text includes support for reverse-slope modes, treats all minor versions together and assumes decisions about handling of ACEs for named users and groups, the relevance of consensus items #26e, #28, and #29 needs to be noted.

This section covers ACL implementations that do not have support for DENY ACEs. In such cases, the absence of support for DENY ACEs forces the ACL to be rewritten in its entirety, rather than have a mode-related section prepended to a mostly unchanged ACL. In addition, the absence of support for DENY ACEs requires special attention to the possible presence of reverse slope modes because OWNER@ is a subset of GROUP@ and both OWNER@ and GROUP@ are subsets of EVERYONE@.

The replacement ACL begins with three ALLOW ACEs for the who values OWNER@, GROUP@, and EVERYONE@. The order in which these are placed in the resultant ACL needs to be adjusted based on the mode value avoid problems with reverse-slope modes. Such problems can arise when a who value processed later contains permission bits not present in previous one so that the later who value, covering a superset of the principals of the earlier one, receives permissions that should not, for example, be granted to the owning user according to the POSIX definition of privileges for the owning group

One way to address this issue is to use ACEs with who values of OWNERNOTGROUP@ and OTHERS@ rather than OWNER@ and EVERYONE@, if appropriate support is available.

In order to address this issue in other case, the three entries need to be sorted in order of descending privilege, as described in Section 10.7.2. As in that case the existence of sets of privilege bits not comparable according to inclusion might force the ACL to be deleted, rather than being replaced by an ACL equivalent to the mode, which in this case cannot exist.

These preliminary ACEs are followed by a series of ACEs derived from the existing ACL with entries copied over or not as described below:

* AUDIT and ALARMS ACEs are copied over.

* ALLOW and DENY ACEs that are marked inherit-only are copied over.

* ALLOW and DENY ACEs that are marked as inheritable without being inherit-only are copied over in a modified form. They need to be marked as inherit only.

* Other ALLOW and DENY ACEs are not copied over. This applies irrespective of the who value, although the reasons for doing this are different for different sorts of who values.

  ACEs with who values of OWNER@, GROUP@, and EVERYONE@ are to be eliminated because they are dealt with in the prepended ACEs.

  ACEs with a who value denoting a specific user or group are to be eliminated because their presence is incompatible with the POSIX file access authorization model.

  ACEs with a special who value (auth-related or miscellaneous) are to be eliminated in order to assume that the file access authorization after setting the mode reflects the mode alone.

10.8. Setting ACL and Not Mode

[Author Aside]: The handling of SHOULD in this section is considered as part of Consensus Item #63d.

When setting the acl or dacl and not setting the mode or mode_set_masked attributes, the permission bits of the mode need to be derived from the ACL. In this case, the ACL attribute SHOULD be set as given. The nine low-order bits of the mode attribute (MODE4_R*, MODE4_W*, MODE4_X*) MUST be modified to match the result of the method in Section 10.5. The three high-order bits of the mode (MODE4_SUID, MODE4_SGID, MODE4_SVTX) SHOULD remain unchanged.

10.9. Setting Both ACL and Mode

When setting both the mode (includes use of either the mode attribute or the mode_set_masked attribute) and the acl or dacl attributes in the same operation, the attributes MUST be applied in the following order: mode (or mode_set_masked), then ACL. The mode-related attribute is set as given, then the ACL attribute is set as given, possibly changing the final mode, as described above in Section 10.8.

10.10. Retrieving the Mode and/or ACL Attributes

[Author Aside]: The handling of SHOULD in this section is considered as part of Consensus Item #63e.

If a server supports any ACL attributes, it may use the ACL attributes on the parent directory to compute an initial ACL attribute for a newly created object. This will be referred to as the inherited ACL within this section. The act of adding one or more ACEs to the inherited ACL that are based upon ACEs in the parent directory’s ACL will be referred to as inheriting an ACE within this section.

Implementors need to base the behavior of CREATE and OPEN depending on the presence or absence of the mode and ACL attributes by following the directions below:

1. If just the mode is given in the call:

   In this case, inheritance SHOULD take place, but the mode MUST be applied to the inherited ACL as described in Section 10.7, thereby modifying the ACL.

2. If just the ACL is given in the call:

   In this case, inheritance SHOULD NOT take place, and the ACL as defined in the CREATE or OPEN will be set without modification, and the mode modified as in Section 10.8.

3. If both mode and ACL are given in the call:

In this case, inheritance SHOULD NOT take place, and both attributes will be set as described in Section 10.9.

4. If neither mode nor ACL is given in the call:

In the case where an object is being created without any initial attributes at all, e.g., an OPEN operation with an opentype4 of OPEN4_CREATE and a createmode4 of EXCLUSIVE4, inheritance SHOULD NOT take place (note that EXCLUSIVE4_1 is a better choice of createmode4, since it does permit initial attributes). Instead, the server SHOULD set permissions to deny all access to the newly created object. It is expected that the appropriate client will set the desired attributes in a subsequent SETATTR operation, and the server SHOULD allow that operation to succeed, regardless of what permissions the object is created with. For example, an empty ACL denies all permissions, but the server need to allow the owner's SETATTR to succeed even though WRITE_ACL is implicitly denied.

In other cases, inheritance SHOULD take place, and no modifications to the ACL will happen. The mode attribute MUST be as computed in Section 10.5, with the MODE4_SUID, MODE4_SGID, and MODE4_SVTX bits clear. If no inheritable ACEs exist on the parent directory, the rules for creating acl, dacl, or sacl attributes are implementation defined. If either the dacl or sacl attribute is supported, then the ACL4_DEFAULTED flag SHOULD be set on the newly created attributes.

10.11. Use of Inherited ACL When Creating Objects

[Author Aside]: The handling of SHOULD in this section is considered as part of Consensus Item #63f.

If the object being created is not a directory, the inherited ACL SHOULD NOT inherit ACEs from the parent directory ACL unless the ACE4_FILE_INHERIT_ACE flag is set.

If the object being created is a directory, the inherited ACL is to inherit all inheritable ACEs from the parent directory, that is, those that have the ACE4_FILE_INHERIT_ACE or ACE4_DIRECTORY_INHERIT_ACE flag set. If the inheritable ACE has ACE4_FILE_INHERIT_ACE set but ACE4_DIRECTORY_INHERIT_ACE is clear, the inherited ACE on the newly created directory MUST have the ACE4_INHERIT_ONLY_ACE flag set to prevent the directory from being affected by ACEs meant for non-directories.

When a new directory is created, the server MAY split any inherited ACE that is both inheritable and effective (in other words, that has neither ACE4_INHERIT_ONLY_ACE nor ACE4_NO_PROPAGATE_INHERIT_ACE set), into two ACEs, one with no inheritance flags and one with ACE4_INHERIT_ONLY_ACE set. (In the case of a dacl or sacl attribute, both of those ACEs SHOULD also have the ACE4_INHERITED_ACE flag set.) This makes it simpler to modify the effective permissions on the directory without modifying the ACE that is to be inherited to the new directory's children.

10.12. Combined Authorization Models for NFSv4.2

The NFSv4 server implementation requirements described in the subsections above apply to NFSv4.2 as well and NFSv4.2 clients can assume that the server follows them.

NFSv4.2 contains an OPTIONAL extension, defined in [RFC8257], which is intended to reduce the interference of modes, restricted by the umask mechanism, with the acl inheritance mechanism. The extension allows the client to specify the umask separately from the mask attribute.

11. Other Uses of Access Control Lists

Whether the Acl or Sacl attributes are used, AUDIT and ALARM ACEs provide security-related facilities separate from the file access authorization provided by ALLOW and DENY ACEs

* AUDIT ACEs provide a means to audit attempts to access a specified file by specified sets of principals.

* ALARM ACEs provide a means to draw special attention to attempts to access specified files by specified sets of principals.

12. ACL_Choice Details

The structure of ACL_Choice attribute is extendible for many of the same reasons that the set of NFSv4 attributes is extendible and shares many of the same mechanisms including use of an easily extended bitmap and a nominally opaque array overlain with a disparate set of individual items. However, these are not exactly the same and the following important differences should be noted:

* The bitmap4 aca_bits is optimized to represent binary choices and has no role in defining the use of the nominally opaque area within aca_dpool.

* The array acs_ditems does define the individual data items that overlay pieces of aca_dpool.

  Because of this structure each individual data item can be located without knowing the length of all preceding ones, as is the case with fattr4.

12.1. Provisions for ACL_Choice Modification

The structure of the ACL_Choice attribute makes it quite easy to add or delete existing flag and data item. However, to ensure interoperability, there are a number of policies set out below, derived from [RFC8178], that restrict how necessary changes can be done.

* New items, whether flags or data item can be done whenever new attributes or similar item can be added. Since all such items are OPTIONAL, defaults need to be specified to cover the case in which the items in question are not present

  This serves a way to address newly discovered behavioral variants.

  In addition to allowing this in extensible minor versions, we will also allow them during the development of this document. New items will be allowed in successive drafts both before and after working group adoption. However, once the document approaches Working Group last call, these will need to cease.

* Deletions, as such, will not be allowed. However, as with OPTIONAL attributes, there will be the opportunity to make existing flags and data items mandatory-to-not-implement in new minor versions, such as, for example NFSv4.3.

It is desirable ability to allow complicated data items by a set of binary choices, when that can be done. In this case the addition of the new flags bits can proceed quickly but the data item to be superseded will remain in the protocol until it can be made mandatory-to-not-implement in a later minor version.

12.2. Storing of ACLs which are not Enforced

The potential ability of servers to accept and store ACEs that they are not prepared to enforce requires the server that supports the ACL_Choice attribute to either:

* Set acd_length to zero in acs_ditems[ACC4IN_STOREUA] to indicate that this functionality is never provided.

* Set acs_ditems[ACC4IN_STOREUA] to refer to an acc4storeua (as described below) within aca_dpool to describe the non-enforced ACEs that they accept.

  Acceptance of ACE4_SYNCHRONIZE is not considered a use of this facility, since that is true of that bit in all cases.

```
<CODE BEGINS>
struct acc4storeua {
    uint32_t    asua_types;
    uint32_t    asua_flags;
    ace4mask    asua_mask:
};
<CODE ENDS>
```

This structure is to be filled in as described below:

* asua_types contain a mask of ACE type that are accepted but not enforced. Each type is represented by a one left-shifted by the numeric value of the type.

* asua_flags contains a mask of ACE flags accepted but and returned on fetched ACLs but not enforced.

  When both ACE4_FILE_INHERIT_ACE and ACE4_DIRECTORY_INHERIT_ACE are set, the is no opportunity to determine potential inheritance so specifying any of ACC4BN_IN1BIT, ACC4BN_INHFULL, or ACC4BN_INHAUTO as part of the ACL_Choice attribute is invalid.

  When one of either ACE4_FILE_INHERIT_ACE or ACE4_DIRECTORY_INHERIT_ACE is set inheritance is limited to being decided by the bit not included if ACC4BN_IN1BIT is set. In addition specifying either of ACC4BN_INHFULL, or ACC4BN_INHAUTO as part of the ACL_Choice attribute is invalid

* asua_mask contains a set of mask bits that the server is prepared to accept and return on fetched ACLs, but have no role in authorization and are not considered for the AUDIT and ALARM functions.

12.3. Provision of Special Accommodations for ACE Mask Bits that Require Support for Windows Semantics

There are a number of cases in which the motivation for a particular ACE mask bit was to accommodate Windows semantics but, for which, as might be expected, NFSv4 was not expected to support the Windows semantics. Given our reluctance to add such major feature as part of a bis document, NFSv4.1 servers may choose to provide for handling of such ACE mask bits in a number of ways.

The handling of ACE4_APPEND_DATA is one such case and there might be others. In this particular case, the way in which the bit was specified, making no direct reference to the Windows Semantics needed, leaving prospective implementers a wide range of choices for a substitute distinction together with the possibility that the missing Windows semantics might be added to NFSv4 in the future.

```
<CODE BEGINS>
enum acc4_app {
               A4A_NONE      = 0,
               A4A_WINONLY   = 1,
               A4A_WINPE     = 2,
               A44_WINPOB    = 3,
               A4W_WINMA     = 4
};
enum acc4_ptype {
               A4PT_ATEND    = 0,
               A4PT_PASTEND  = 1,
               A4PT_PEPH     = 2
};
union acc4_winsa switch(acc4_app how) {
 case A4A_WINPOB:
 case A4A_WINMA:
               acc4_ptype    a4wsa_pt;
 default:
               void;
};
<CODE ENDS>
```

The union acc4_winsa overlays the formally opaque region of aca_dpool selected by acs_ditems[ACC4IN_WINSA].

Within acc4_winnsa, the acc4_app how, selects the possible ways that the distinction between writing data and appending data is realized by the server for the current filesystem:

A4A_NONE Indicates that the distinction is not made

A4A_WINONLY Indicates that the distinction is not made for NFSv4 WRITEs but is available for IO to NFSv4 files that is performed in other ways.

This would make the bit's use like that of ACE4_SYNCHRONIZE in which the bit is available only for non-NFSv4 use (e.g., for use by local Windows-oriented file access or another remote file access protocol such as SMB.

A4A_WINPE Indicates that the distinction is only made for non-NFSv4 use and for the use of Windows-equivalent semantics added to NFSv4.

The latter would require NFSv4 extension along the lines discussed in Appendix F.3.

A4A_WINPOB Indicates that this distinction is made in two cases:

* When windows is used, as described in case of A4A_WINONLY above.

* When a client is using a version of NFS which does not support the Windows-like enhancements to support append-only WRITEs.

  This include NFSv4, NFSv4.0, and NFSv4.1 clients, together for later minor versions in which the extensions might be defined while the server does not support them.

  In this case, the details of the support for the distinction are based on the offset and length of the WRITE and applied as specified by the acc4_ptype in a4wsa_pt. In this case, the implementers need to be aware that:

  - The distinction needs to be made WRITE time rather than at OPEN time.

  - In some cases of acc4_ptype, there are WRITES that cross the end-of-file and require both WRITE and APPEND mask bits

A4A_WINMA Indicates that this distinction is made in three cases:

* When windows is used, as described in case of A4A_WINONLY above.

* When Windows-equivalent semantics added to an NFSv4 minor version and the sever supports use of those semantic.

* When a client is using a version of NFS which does not support the Windows-like enhancements to support append-only WRITEs.

  This include NFSv4, NFSv4.0, and NFSv4.1 clients, together for later minor versions in which the extensions might be defined while the server does not support them.

  In this case as well. the details of the support for the distinction are based on the offset and length of the WRITE and applied as specified by the acc4_ptype in a4wsa_pt. In this case, the implementers need to be aware that:

  - The distinction needs to be made WRITE time rather than at OPEN time.

  - In some cases of acc4_ptype, there are WRITES that cross the end-of-file and require both WRITE and APPEND mask bits

For the final two cases in which the distinction is made at WRITE time, the specifics of the distinction are represented as an acc4_ptype. Despite the fact that all the possible choices below ensure that A WRITE operation done by a principal without the WRITE ACE mask cannot modify existing data, it may be helpful to the client to know which of the following is being used by the server.

A4PT_ATEND Writes are classified by using the start point with only those extending the file with no gap considered as appending

While this is natural, it treat many writes hat have no chance of modifying existing bytes as rejectable since they are no appends, erroneously treating them as if they modify existing bytes

A4PT_PASTEND Writes are classified by using the start point with only those extending the file considered as appending.

While this is a better choice, it is still at variance with previous discussions which assumed that a non-appending WRITE should be considered as overwriting existing bytes, despite the fact that this is not always true.

A4PT_PEH Writes are classified by using the start point with only those extending the file considered as appending. in addition, a special allowance is made for WRITES to previously unwritten areas, i.e., holes.

Clients aware of this could set up separate regions containing multiple separate sequential logs, when multiple clients are writing and they cannot be restricted using sharing options or delegations.

12.4. ACL_Choice Reporting of ACL Size Limits

[Consensus Needed (Item #120c), Through end of section]: Although the XDR definitions of the acl, sacl, and dacl attributes contain variable-length ACE arrays, there likely to be limits on the sizes of these arrays imposed by the filesystem. These values are made available to client as part of the ACL_Choice attribute, within a data item of type IN_ASLIM.

```
<CODE BEGINS>
struct as4lim {
        uint32_t    asl_limguar;
        uint32_t    asl_limrout;
        uint32_t    asl_limfetch;
};
struct as4ltype {
        as4lim      alt_lim;
        acetype4    alt_type;
};
struct acc4_aslim
        as4lim      a4as_acl;
        as4lim      a4as_sacl;
        as4lim      a4as_dacl;
        as4ltype    a4as_types<>;
};
<CODE ENDS>
```

The as4lim structure provides size limit information applicable to a particular attribute or ace type. It consists of the following elements:

* asl_limguar is a the limit guarantee which is a count of ACEs such that ACLs containing that number of ACEs or fewer are sure to be small enough for the filesystem to successfully store.

* asl_limrout is the routine limit. This limit can higher than the guarantee and makes allowance for occasional situations in which certain ACLs cannot be stored either because they contain ACEs with unusually large space demands or because a fixed ACE space is shared with storage used for other features.

* asl_limfetch is a limit for the size of ACL-related attributes to be fetched using GETATTR. It is specified in terms of 32-bit XDR words and applies to the output provides including XDR padding and associated overhead.

  When the value is zero it can be assumed that the filesystem imposes no such limit

The as4ltype structure provides limits applicable to ACEs of a specific type. It consists of the following elements:

* alt_lim is a set of limits applicable only to ACEs of the specified type.

* alt_type is the type that this entry is applicable. The type is expressed as an acetype4 which is the identifier appearing in the ACE itself (e.g. ACE4_ACCESS_ALLOWED_ACE_TYPE) and can include additional ACE types if they are added by future extensions.

The acc4_aslim structure overlays the selected data item. Its elements are described below:

* The elements a4as_{acl, sacl, and dacl} provide, for each supported ACL-related attribute, information about the size limits for that attribute. For unsupported attribute, all fields within the corresponding as4lim are set to zero.

  When acl and at lest one other ACL-related attribute is supported there are a number of constraints that derive from the inclusion of dacl and sacl within acl. For each field FLD within {asl_limguar, asl_limrout, asl_limfetch}, acl.FLD >= MAX(dacl.FLD, sacl.FLD because the acl attribute includes the contents of both sacl and dacl. In addition acl.FLD <= dacl.FLD + sacl.FLD, although it is common for these two to be equal

* To support cases in which file systems have limitations on the number of ACEs of particular types, the array a4as_types provides for each limited ACE type, the type and the associated limit.

  Types for which such limits do not exist will not be mentioned in the array. As a result, when no such limits exist, this array is of length zero.

12.5. Advice/Recommendations Regarding ACL_Choice

[Consensus Needed (Items #12e, #105v, #117k), For entire section]:

The ACL_Choice attribute allows the clients to be informed about behavioral choices the server is allowed to make. The fact that the server is allowed to make these choices and report them does not necessarily imply that these choices are valid. They may be allowed solely because they always have been allowed since previous specifications chose to be quite lax in these matters. Also, to correct situations in which previous specifications gave servers undue latitude in making such choices, some behavior previously not felt to be noteworthy is recommended against using "SHOULD" or "SHOULD NOT" with the understanding that the valid reasons to bypass the recommendation are limited to the reliance on earlier specifications.

In the table below, we summarize the existing items defined within the ACL_Choice attribute, including both flags and data items, providing summary advice/recommendations regarding their use together with some expectations/speculations regarding future development. The advice is summarized using the codes listed below.

Vox Valid OPTIONAL extension.

An extension to the core that was allowed to be present or not according to previous specifications, without clearly stating that. The absence of the extension was treated as an excusable implementation flaw.

Upx Valid OPTIONAL extension useful for the support of draft-POSIX ACLs.

An extension to the core that was allowed to be present or not according to previous specifications, without clearly stating that. Although the absence of the extension was treated as an excusable implementation flaw, it is useful for supporting draft-POSIX ACLs.

Npx Valid OPTIONAL extension necessary for the support of draft-POSIX ACLs.

An extension to the core that was allowed to be present or not according to previous specifications, without clearly stating that. Although he absence of the extension was treated as an excusable implementation flaw, it is necessary for supporting draft-POSIX ACLs.

Vax Valid absence of an OPTIONAL extension

A choice to support core functionality without any of a set of extensions that were allowed to be present or not according to previous specifications. Only the core functionality is provided despite the absence of a clear statement that this was REQUIRED.

Lux Valid OPTIONAL extension with limited uses

Mentioned in passing as acceptable in previous specifications. Despite the fact that it should avoided because of its unfortunate side-effects, it has some valid use cases.

Vbc Valid Behavioral Choice

An extension to the core that was allowed to be present or not according to previous specifications.

Nbc Needed Behavioral Choice.

Whether allowed by previous specifications or not, needed by clients to provide needed functionality.

Dbc Dubious Behavioral Choice.

Whether allowed by previous specifications or not, an acknowledged behavioral variant for which substantive justification is hard to perceive.

Sbt Should be True.

Indicate that setting this false has negative consequences of which clients and servers need to be aware.

Sbf Should be false.

Indicate that setting this true has negative consequences of which clients and servers need to be aware.

Mbc Miscellaneous behavioral choices.

Describes a set of matters for which previous specifications gave no behavioral guidance allowing a range of server behaviors to be chosen.

Item	Adv.	Future Dev.
ACC4BN_NEINGM	Vbc	Convergence desirable but unlikely. Should work to allow client choice.
ACC4BN_SEPFWX	Vox	Will be guided by client needs
ACC4BN_SEPAFD	Vox	Will be guided by client needs
ACC4BN_SEPDE	Vox	Will be guided by client need
ACC4BN_RNASDI	Sbf	Old approach might or might not be implemented
ACC4BN_NAD	Vox	Will be guided by client needs
ACC4BN_RNADMOD	Sbt	Old approach makes no sense. Will need to look at implementations.
ACC4BN_MBCA	Sbf	Would like to get rid of this when we can.
ACC4BN_SMJUST3	Sbf	Might be possible to get rid of this.
ACC4BN_SMOLD	Sbf	Would like to get rid of this but can’t.
ACC4BN_SMFULL	Sbt	Want to get here. Would take a while.
ACC4BN_3MASKB	Vax	No changes expected.
ACC4BN_AUTHWHO	Vbc	Might not find any implementations.
ACC4BN_INNO	Dbc	Need more than this level of inheritance to support draft-POSIX ACLs
ACC4BN_IN1BIT	Upx	Need at least this level of inheritance to support draft-POSIX ACLs
ACC4BN_INHFULL	Upx	Sometimes needed for support of draft-POSIX ACLs.
ACC4BN_INHAUTO	Vox	Not sure any implementations exist.
ACC4BN_AACPS	Upx	Helpful in providing draft-POSIX ACL authorization semantics.

ACC4BN_OTHWHO	Dbc	Special who values with unclear denotation.
ACC4BN_DPWHO	Vox	Special who values to deal with reverse-slope modes when DENY ACEs are not supported.
ACC4BN_RVINV	Sbf	Would like to get rid of these.
ACC4BN_SVTX	Vbc	Should be quite common. Usable even if ACLs not supported.
ACC4BN_SVTXOLD	Sbf	Would like to get rid of these.
ACC4IN_OWNMB	Vox	Expect to be common. If they are all the same, can get rid of this.
ACC4IN_ODDMB	Mbc	Will try to simplify over time.
ACC4IN_STOREUA	Lux	Expect some implementations but have not seen any.

Table 6: Table of ACL_Choice Items (with Advice)

13. Security Considerations

There are very few Security considerations specific to this document. Security considerations for NFSv4 as a whole are dealt with in the Security Considerations section of [I-D.dnoveck-nfsv4-security].

In the definition of attributes with important security consequences, it is important that the security framework be well understood so that clients can understand the expected security consequences of choices made in setting ACLs. Given this need, the difficulty of implementing particular choices should not allow something approaching the required to be accepted even without implementing the required semantics. In such situations, it is best to treat difficult-to-implement feature as explicitly OPTIONAL, as has been done in this approach to ACL description.

This document has had to deal with the fact that earlier specifications did not follow the advice above, with the result that implementers, following existing Standards- Track specifications, were led to implementations that now appear misguided. Rather than retrospectively declare such implementations non-compliant, the document has recommended the implementation now felt to suitable using "SHULD" wile treating the reliance on a previous misguided specification as a valid reason to bypass the recommendation.

The implementation and use of the ACL_Choice attribute provides a way for clients to determine what aspects of a server’s implementation can be relied upon when it request ACL features beyond a small core referred elsewhere as "UNIX ACLs"

14. IANA Considerations

This document requires no actions from IANA>

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", RFC 8178, DOI 10.17487/RFC8178, July 2017, <https://www.rfc-editor.org/info/rfc8178>.

[RFC7530] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", RFC 7530, DOI 10.17487/RFC7530, March 2015, <https://www.rfc-editor.org/info/rfc7530>.

[RFC7531] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 External Data Representation Standard (XDR) Description", RFC 7531, DOI 10.17487/RFC7531, March 2015, <https://www.rfc-editor.org/info/rfc7531>.

[RFC8881] Noveck, D., Ed. and C. Lever, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 8881, DOI 10.17487/RFC8881, August 2020, <https://www.rfc-editor.org/info/rfc8881>.

[I-D.ietf-nfsv4-rfc8881bis]
Noveck, D., "Network File System (NFS) Version 4 Minor Version 1 Protocol", Work in Progress, Internet-Draft, draft-ietf-nfsv4-rfc8881bis-04, 17 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nfsv4-rfc8881bis-04>.

[I-D.dnoveck-nfsv4-rfc5662bis]
Noveck, D., "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", Work in Progress, Internet-Draft, draft-dnoveck-nfsv4-rfc5662bis-06, 23 May 2025, <https://datatracker.ietf.org/doc/html/draft-dnoveck-nfsv4-rfc5662bis-06>.

[I-D.dnoveck-nfsv4-security]
Noveck, D., "Security for the NFSv4 Protocols", Work in Progress, Internet-Draft, draft-dnoveck-nfsv4-security-13, 16 November 2025, <https://datatracker.ietf.org/doc/html/draft-dnoveck-nfsv4-security-13>.

15.2. Informative References

[RFC3010] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "NFS version 4 Protocol", RFC 3010, DOI 10.17487/RFC3010, December 2000, <https://www.rfc-editor.org/info/rfc3010>.

[RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, DOI 10.17487/RFC3530, April 2003, <https://www.rfc-editor.org/info/rfc3530>.

[RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

[RFC8276] Naik, M. and M. Eshel, "File System Extended Attributes in NFSv4", RFC 8276, DOI 10.17487/RFC8276, December 2017, <https://www.rfc-editor.org/info/rfc8276>.

[RFC9754] Haynes, T. and T. Myklebust, "Extensions for Opening and Delegating Files in NFSv4.2", RFC 9754, DOI 10.17487/RFC9754, March 2025, <https://www.rfc-editor.org/info/rfc9754>.

[I-D.ietf-nfsv4-posix-acls]
Macklem, R., "POSIX Draft ACL support for Network File System Version 4, Minor Version 2", Work in Progress, Internet-Draft, draft-ietf-nfsv4-posix-acls-01, 8 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nfsv4-posix-acls-01>.

[Gr端nbacher]
Gr端nbacher, A., "POSIX Access Control Lists on Linux", USENIX 2003 Annual Technical Conference Proceedings, June 2003.

Appendix A. Document History and Associated Expectations

This appendix will provide a review of the pre-adoption history of this document and serve to focus attention on recent and forthcoming changes to the document necessary to move toward Working Group Last Call.

A.1. Document Progress Before Adoption

During the pre-adoption history of this document which consisted of a number of drafts of draft-dnoveck-nfsv4-acls, a number of attempts were made to address the problems discussed in Section 3 without requiring any changes in existing implementations while still keeping open a clear path to interoperable implementations of:

* ACLs in the form described by previous version specifications that assumed the Necessary Windows semantics that motivated the adoption of the Windows ACL model as a basis for NFSv4 ACLs.

  Despite this need, this was never provided due to a pervasive unwillingness to deal with the specification of authorization semantics. In addition, the lack of participation of Window-knowledgeable Working Group members together with the lack of commonly available API descriptions comparable to those available for POSIX meant that decades went by without this gap being addressed.

* ACLs semantically compatible with a POSIX approach to authorization were needed because most NFSv4 clients and servers were written in a compatible style and would have great difficulties in shifting to a non-POSIX model.

While earlier minor version specifications assumed that the draft-POSIX ACL model, because it used a coarser-grained permission could be considered to be a sub-model of the NFSv4, it became clear that this was not correct and only a small subset of that, referred to elsewhere as UNIX ACLs, could serve in that role.

Although attempts were made to expand the NFSv4 ACL model to provide for the potential inclusion of draff-POSIX ACL semantics, these attempts were not successful. Instead, it was decided that supporting draft-POSIX ACLs was best done by defining new ACL attributes for that purpose as is done in [I-D.ietf-nfsv4-posix-acls]

As a result of the decision to provide support for draft-POSIX ACLs by a completely separate mechanism, the job of this document became simpler in a number of ways:

* We avoided having to deal with two completely different approaches to combining the results of individual ACE checks and to the handling of ACL inheritance.

  Attempts to do so had resulted in hard-to-understand combined ACL model.

* Since the conceptual basis of the ACL, the Windows could be treated as OPTIONAL extensions.

  This superseded the previous approach in which windows ACLs were the conceptual basis of the feature and UNIX ACLs were addressed by lax approach to specification which needed to be undone in order to provide a useful security feature.

* The addition of the ACL_Choice OPTIONAL attribute provided a way for clients needing Windows semantic could determine whether they were provided by the server. Without that ability, the appearance of these feature in the spec were ineffective, since servers need not implement them, and clients could not determine whether a server had implemented them, undercutting any motivations servers might have for providing an implementation.

  The addition of a new feature is normally beyond the expected scope of a bis document. However, in the case, it is justified by [RFC8178] which allows such OPTIONAL to correct protocol "defects", which this feature uncertainty certainly is.

This shift left us with the following doable tasks:

* Clarify the semantics of Windows features that never got a clear description.

  An important case is the classification of WRITES into those cases covered by the two ACE mask bits for WRITE and APPEND.

  This requires a higher level of participation by Windows-knowledgeable people than was available during the specification of the ACLs feature.

* Clearly addressing other semantic gaps in Features outside the POSIX model.

* Making clear the limits of UNIX ACLs ad the reasons the NFSv4 ACL model does not support draft-POSIX ACLs

  More explanation is required regarding the differences and incompatibilities of the two models with regard to ACE check result combination and ACL inheritance.

As a consequence, our work in subsequent drafts will involve:

* Work to address the gaps cited in the list immediately above.

* An effort to resolve existing consensus items now described in Appendix C, focusing first on those connected with effecting the conceptual changes that were made before document adoption as described at the start of this section.

A.2. Changes Made in Draft -01

A major focus of this draft is to clearly embrace the revised approach to the problem in the original specifications of ACLs by making it clear that draft-POSIX will dealt with using a separate feature. These changes include:

* A major revision to the Abstract to present the approach chosen and eliminate text that was more appropriate when that decision had not been made.

  As part of that revision, a paragraph marked to be removed upon RFC publication was removed since this document is now with some possible modification to be published as an RFC.

* Adding more discussion of Windows semantic requirements to the Introduction (Section 1) to reflect the need to clearly describe these features despite the fact they are no longer considered the core of the new ACL model.

* Creation of a new Appendix A.1 explaining how we arrived at the current approach.

In addition the following additional changes were made:

* Changes to reflect the conversion of rfc5661bis to rfc8881bis.

* Creation of a new Appendix A dealing with the document history and near-term expectations for further development.

* Making it clear in Section 8.3.6 that further clarification of the distinction between the WRITE and APPEND ACE mask bits with details of issues to be supplied later.

  Adding a new Section 8.3.7 to make clear the questions that need to be resolved about this distinction. In addition, because of the possibility of incompatible implementation, ACL_Choice needs to be enhanced to provide information about possible server implementations. These are discussed below.

* A number of important changes were made to the ACL_Choice description that arose as part of the need to provide information about implementation of the WRITE/APPEND distinction.

* Some work on potential extensions including the addition of a new Appendix F.3 and some additions to Appendix F.1.

* There was extensive work done on consensus items as described in Appendix C.2.4. Tis includes revisions of issue states and the additional classification undertaken in Appendix C.3

* Significant additions to the Acknowledgments.

A.3. Changes Made in Draft -02

The following changes, many of which ere previously planned as described in Appendix A.3 of draft -01, have been effected:

* Insertion of markers where they were missing in sourcecode sections

* Further work on early sections (i.e., Abstract through Section 3) to put forth the new approach to ACL description forthrightly, isolating explanation of the previous problems with ACL and the choices leading up to the current approach in separate sections.

This includes a more focused description of the problems in previous approaches in a revised Section 3 and a new section, with a focus on the changes in approach and their benefits.

* Extensive revisions of Section 10 to eliminate sections marked "(vestigial)" and "(discussion)" dealing with these matters in Section 3 and the new section dealing with approach changes.

  Sections marked "(proposed) will lose that designation and will be reworded to reflect the new treatment with any uncertainty represented in the form of Consensus Issues.

* Further analysis of Consensus Issues marked Delay-Likely in Appendix C.3.

  One important focus will be clearer discussion of where multiple issues are related or need to be addressed together with consensus issues in other documents.

  Another important focus is better characterization of the possible alternatives for issues where this is a valid option.

* As part of the cleanup of Consensus Item #5, added discussion of possible extension regarding ACCESS recordable events to Appendix D.2.

* Creation of a more substantial Security Considerations section in Section 13, which argues that clear definitions of authorization semantics is an important Security Consideration in the case of ACLs.

In addition, the following changes, connected to the handling of large ACLs, have been made.

* A new section (S 5.5, explaining the subject of ACL size limits has been created.

* A new section (S 5.3 has been created. It deals with errors to be added in connection with exceeding ACL size limits.

* A new section (S 12.4, dealing with the reporting of ACL size limits, has been created.

* Creation of a new Appendix D.2.2 dealing with possible ACL_Choice extensions to deal with ACL size limits more completely.

  This is part of a general reorganization of Appendix D.2 including creation of Appendix D.2.1.

* Creation of a new Appendix F.4 to deal with possible extensions to support Massive ACLs.

A.4. Changes Made in Draft -03

The following changes, made in draft -03, bring the NFSv4 ACL model into closer alignment with Windows semantics.

* Addition of ACE4_{READ,WRITE}_EXT_ATTRS for authorization related to extended attributes.

* Addition of ACE4_SACL_ACCESS to monitor use of and change to AUDIT and ALARM ACEs.

* Preliminary discussion of a possible extension for support of Massive ACLs (and similar attribute).

* Addition of support for CREATOR_OWNER@ and CREAOTR_GROUP.

* Addition of support for OWNER_RIGHTS@.

An important set of changes concern changes to the coordination of Sections 8.3 and ‘ 10.

* Creation of a new Section 8.3.1 devoted to the initial mask bit

* Restructuring Section 8.2 in draft-02 into Sections 8.3.14 through 8.3.16

* Changes to discussion of setting mode and computing mode in Section 10 to use the arrays MaskFromPrivVal and MaskBitsNeeded.

In addition, we made the following related changes:

* Created Section 8.3.8 to deal more coherently for finer-grained Access mask bit tied to the ability to read a file (including reading for execute).

* Explicitly noted the existence of the gaps we are filling in this draft.

Another important set of changes concerns the handling of "automatic inheritance".

* Creation of a new Section 8.2 dealing with ACE inheritance in general, including both server-effected inheritance and client-managed inheritance, each dealing with its own subsection.

This parallel is part of a synchronic description OF INHERITANCE replaces the previous handling of automatic inheritance as a separate feature which it was in NFSv4.1 but is no longer appropriate to emphasize in an NFSv4-wide document.

* Transfer of the former Section 9 dealing with the specifics of automatic inheritance to Section 8.2.3

* Update of Sections 5.10 and 5.11 to clarify their relationship to automatic inheritance.

* Addition of an exploration of possible equivalents for automatic inheritance to draft-POSIX ACLs as part of a reorganized Appendix E.

A.5. Moving Document Forward

Given the current state of the document, our work in drafts leading toward an eventual WGLC will be of the following kinds:

* Possible discovery of and response to previously unrecognized issues, discovered as result of document review.

* Conversion of issues to a terminal state and the consequent document changes once the issue is presented to the Working Group and a consensus established.

* Work to identify and resolve differences of opinion, primary for those issue now considered as Delay-Likely.

Appendix B. Vestigial and Transitional Text

B.1. Handling of Deletion (Vestigial)

[Author Aside]: This section contains the former content of Section 8.3.20. All unannotated paragraphs within it are to be considered the Previous Treatment associated with consensus item #12d.

[Author Aside, Including List]: Listed below are some of the reasons that I have tried to replace the existing treatment rather than address the specific issues mentioned here and in later asides.

* The fact that there is no clear message about what servers are to do and about what behavior clients might rely on. This derives in turn from the use of "SHOULD" in contexts in which it is clearly not appropriate, combined with non-normative reports of what some

systems do, and the statement that the approach suggested is a way of providing semantics rather unhelpfully describe as "something like traditional UNIX-like semantics".

* The complexity of the approach without any indication that there is a need for such complexity makes me doubtful that anything of this sort was actually implemented, especially since the text is so wishy-washy about the need for server implementation. The probability that it would be implemented so widely that clients might depend on it is even more remote.

* The fact that how audit and alarm issues are to be dealt with is not addressed at all.

* The fact that this treatment makes the most obvious potential use of ACE4_DELETE, to deny a user the ability to delete a file, essentially unusable.

* The fact that the previous treatment creates a huge security hole, by allowing deletion of directory entries in case in which the user requesting deletion does not have ownership or any sort of write permission on the directory being modified.

Two access mask bits govern the ability to delete a directory entry: ACE4_DELETE on the object itself (the "target") and ACE4_DELETE_CHILD on the containing directory (the "parent").

Many systems also take the "sticky bit" (MODE4_SVTX) on a directory to allow unlink only to a user that owns either the target or the parent; on some such systems the decision also depends on whether the target is writable.

[Author aside]: Beyond the fact that this observation is a non-normative report of server behavior, it is worth noting that the motivation for this, to allow users to delete their own files within a shared directory, is vitiated by the following paragraph.

Servers SHOULD allow unlink if either ACE4_DELETE is permitted on the target, or ACE4_DELETE_CHILD is permitted on the parent. (Note that this is true even if the parent or target explicitly denies one of these permissions.)

[Author aside]: I don’t understand what possible justification might allow a directory to be modified when the request is not the owner and has no write permission for the directory being *modified*. In any case no justification is offered. Similarly, this allows deletion on a file even when the owner has denied a user the right to delete it. It is hard to figure out why.

If the ACLs in question neither explicitly ALLOW nor DENY either of the above, and if MODE4_SVTX is not set on the parent, then the server SHOULD allow the removal if and only if ACE4_ADD_FILE is permitted.

[Author Aside]: Unclear what SHOULD means here and the use of ACE4_ADD_FILE is clearly wrong (should be ACE4_DELETE_CHILD).

In the case where MODE4_SVTX is set, the server may also require the remover to own either the parent or the target, or may require the target to be writable.

[Author Aside]: "may also require" differs from how this is documented for UNIX in general, where the ownership replaces the test for write privilege on the directory.

This allows servers to support something close to traditional UNIX-like semantics, with ACE4_ADD_FILE taking the place of the write bit.

[Author Aside]: Given that ACLs might or might not be present, "something close" (whatever that means) is not enough, since the ACL and non-ACL cases need to work together seamlessly, so as to provide something implementable and understandable.

B.2. Computing a Mode Attribute from an ACL (vestigial)

[Previous Treatment (Item #27c)]: The following method can be used to calculate the MODE4_R*, MODE4_W*, and MODE4_X* bits of a mode attribute, based upon an ACL.

[Author Aside]: "can be used" says essentially "do whatever you choose" and would make Section 10 essentially pointless. Would prefer "is to be used" or "MUST", with "SHOULD" available if valid reasons to do otherwise can be found.

[Author Aside, Including List]: The algorithm specified below, now considered the Previous Treatment associated with Item #24a, has an important flaw in does not deal with the (admittedly uncommon) case in which the owner_group has less access than the owner or others have less access than the owner-group. In essence, this algorithm ignores the following facts:

* That GROUP@ includes the owning user while group bits in the mode do not affect the owning user.

* That EVERYONE includes the owning group while other bits in the mode do not affect users within the owning group.

[Previous Treatment (Item #28b)]: First, for each of the special identifiers OWNER@, GROUP@, and EVERYONE@, evaluate the ACL in order, considering only ALLOW and DENY ACEs for the identifier EVERYONE@ and for the identifier under consideration. The result of the evaluation will be an NFSv4 ACL mask showing exactly which bits are permitted to that identifier.

[Previous Treatment (Item #28b)]: Then translate the calculated mask for OWNER@, GROUP@, and EVERYONE@ into mode bits for, respectively, the user, group, and other, as follows:

B.3. Alternatives in Computing Mode Bits (vestigial)

[Author Aside]: All unannotated paragraphs within this section are to be considered the Previous Treatment corresponding to Consensus Item #27d.

Some server implementations also add bits permitted to named users and groups to the group bits (MODE4_RGRP, MODE4_WGRP, and MODE4_XGRP).

Implementations are discouraged from doing this, because it has been found to cause confusion for users who see members of a file’s group denied access that the mode bits appear to allow. (The presence of DENY ACEs may also lead to such behavior, but DENY ACEs are expected to be more rarely used.)

[Author Aside]: The text does not seem to really discourage this practice and makes no reference to the need to standardize behavior so the clients know what to expect or any other reason for providing standardization of server behavior.

The same user confusion seen when fetching the mode also results if setting the mode does not effectively control permissions for the owner, group, and other users; this motivates some of the requirements that follow.

[Author Aside]: The part before the semicolon appears to be relevant to Consensus Item #23 but does not point us to a clear conclusion. The statement certainly suggests that the nine low-order bits of the mode select one of 512 corresponding ACLs is a desirable but the absence significant details or a more direct statement to that effect suggest that this is a server implementer choice.

[Author Aside]: The part after the semicolon is hard to interpret in that it is not clear what "this" refers to or which requirements are referred to by "some of the requirements that follow". The author would appreciate hearing from anyone who has insight about what might have been intended here.

B.4. Setting Mode and not ACL (vestigial)

[Author Aside]: All unannotated paragraphs are to be considered the Previous treatment of Consensus Item #30a.

When any of the nine low-order mode bits are subject to change, either because the mode attribute was set or because the mode_set_masked attribute was set and the mask included one or more bits from the nine low-order mode bits, and no ACL attribute is explicitly set, the acl and dacl attributes must be modified in accordance with the updated value of those bits. This must happen even if the value of the low-order bits is the same after the mode is set as before.

Note that any AUDIT or ALARM ACEs (hence any ACEs in the sacl attribute) are unaffected by changes to the mode.

In cases in which the permissions bits are subject to change, the acl and dacl attributes MUST be modified such that the mode computed via the method in Appendix B.2 yields the low-order nine bits (MODE4_R*, MODE4_W*, MODE4_X*) of the mode attribute as modified by the attribute change. The ACL attributes SHOULD also be modified such that:

1. If MODE4_RGRP is not set, entities explicitly listed in the ACL other than OWNER@ and EVERYONE@ SHOULD NOT be granted ACE4_READ_DATA.

2. If MODE4_WGRP is not set, entities explicitly listed in the ACL other than OWNER@ and EVERYONE@ SHOULD NOT be granted ACE4_WRITE_DATA or ACE4_APPEND_DATA.

3. If MODE4_XGRP is not set, entities explicitly listed in the ACL other than OWNER@ and EVERYONE@ SHOULD NOT be granted ACE4_EXECUTE.

Access mask bits other than those listed above, appearing in ALLOW ACEs, MAY also be disabled.

Note that ACEs with the flag ACE4_INHERIT_ONLY_ACE set do not affect the permissions of the ACL itself, nor do ACEs of the type AUDIT and ALARM. As such, it is desirable to leave these ACEs unmodified when modifying the ACL attributes.

Also note that the requirement may be met by discarding the acl and dacl, in favor of an ACL that represents the mode and only the mode. This is permitted, but it is preferable for a server to preserve as much of the ACL as possible without violating the above requirements. Discarding the ACL makes it effectively impossible for a file created with a mode attribute to inherit an ACL (see Section 10.11).

B.5. Setting Mode and not ACL (Discussion)

[Author Aside]: All unannotated paragraphs are to be considered Author Asides relating to Consensus Item #30b.

Existing documents are unclear about the changes to be made to an existing ACL when the nine low-order bits of the mode attribute are subject to modification using SETATTR.

A new treatment needs to apply to all minor versions. It will be necessary to specify that, for all minor versions, setting of the mode attribute, subjects the low-order nine bits to modification.

One important source of this lack of clarity is the following paragraph from Appendix B.4, which we refer to later as the trivial-implementation-remark".

> Also note that the requirement may be met by discarding the acl and dacl, in favor of an ACL that represents the mode and only the mode. This is permitted, but it is preferable for a server to preserve as much of the ACL as possible without violating the above requirements. Discarding the ACL makes it effectively impossible for a file created with a mode attribute to inherit an ACL (see Section 10.11).

The only "requirement" which might be met by the procedure mentioned above is the text quoted below.

> In cases in which the permissions bits are subject to change, the acl and dacl attributes MUST be modified such that the mode computed via the method in Appendix B.2 yields the low-order nine bits (MODE4_R*, MODE4_W*, MODE4_X*) of the mode attribute as modified by the attribute change.

While it is true that this requirement could be met by the specified treatment, this fact does not, in itself, affect the numerous recommendations that appear between the above requirement and the trivial-implementation-remark.

It may well be that there are implementations that have treated the trivial-implementation-remark as essentially allowing them to essentially ignore all of those recommendations, resulting in a situation in which were treated as if it were a trivial-implementation-ok indication. How that issue will be dealt with in a replacement for Appendix B.4 will be affected by the working group's examination of compatibility issues.

The following specific issues need to be addressed:

* Beyond the possible issues that arise from the trivial-implementation-ok interpretation, the treatment in Appendix B.4, by pointing specifically to existing INHERIT_ONLY ACEs obscures the corresponding need to convert ACE's that specify both inheritance and access permissions to be converted to INHERIT_ONLY ACEs.

* Reverse-slope modes

  The effect of ignoring this case is often so pervasive that the algorithms offered cannot be patched to avoid the issue but need to be rethought.

* Named users and groups.

  The particular handling of these in computing mode, could conceivably affect other aspects of mode handling as well.

  We will need to consider the behavior of clients and servers to get a better handle on these issues.

* The exact bounds of what within the ACL is covered by the low-order bits of the mode.

  A particular concern is the handling of ACE mask bits that are neither derived directly from a POSIX permission bit nor control a subset of the actions controlled by a POSIX permission bit. It is often assumed in previous specification that no such bits exist but, since that is not the case, the issue needs to be addressed somehow.

We have wound up accommodating a large set of bits, but might need to revisit this issue if and when we decide to standardize the handling of mask bits that are not finer-grained version of one of the three POSIX permission bits.

It appears that for many of the issues, there are many possible readings of the existing specs, leading to the possibility of multiple inconsistent server behaviors. Furthermore, there are cases in which none of the possible behaviors described in existing specifications meets the needs.

As a result of these issues, the existing specifications do not provide a reliable basis for client-side implementations of the ACL feature which a Proposed Standard is normally expected to provide.

Appendix C. Issues for which Consensus Needs to be Ascertained

C.1. List of Issues

This section helps to keep track of specific changes which the author has made or intends to make to deal with ACL-related issues that appear in RFCs 7530 and 8881. The changes listed here exclude those which are clearly editorial but includes some that the author believes are editorial but for which the issues are sufficiently complicated that working group consensus on the issue is probably necessary.

These changes are presented in the table below, organized into a set of "Consensus Items" identified by the numeric code appearing in annotations in the proposed document text. For each such item, a type code is assigned with separate sets of code define for pending items and for those which are no longer pending.

The following codes are defined for pending consensus items:

* "NM" denotes a change which is new material that is not purely editorial and thus requires Working Group consensus for eventual publication.

* "BE" denotes a change which the author believes is editorial but for which the change is sufficiently complex that the judgment is best confirmed by the Working Group.

* "BC" denotes a change which is a substantive change that the author believes is correct. This does not exclude the possibility of compatibility issues becoming an issue but is used to indicate that the author believes any such issues are unlikely to prevent its eventual acceptance.

* "CI" denotes a change for which the potential for compatibility issues is a major concern with the expected result that working group discussion of change will focus on clarifying our knowledge of how existing clients and server deal with the issue and how they might be affected by the change or the change modified to accommodate them.

* "NS" denotes a change which represents the author's best effort to resolve a difficulty but for which the author is not yet confident that it will be adopted in its present form, principally because of the possibility of troublesome compatibility issues.

* "NE" denotes change based on an existing issue in the spec but for which the replacement text is incomplete and needs further elaboration.

* "WI" denotes a potential change based on an existing issue in the spec but for which replacement text is not yet available because further working group input is necessary before drafting. It is expected that replacement text will be available in a later draft once that discussion is done.

* "LD" denotes a potential change based on an existing issue in the spec but for which replacement text is not yet available due to the press of time. It is expected that replacement text will be available in a later draft.

* "EV" denote a potential change which is tentative or incomplete because further details need to be provide or because the author is unsure that he has a correct explanation of the issue. It is expected that replacement text will be available in a later draft.

The following codes are defined for consensus items which are no longer pending.

* "RT" designates a former item which has been retired, because it has been merged with another one or otherwise organized out of existence.

  Such items no longer are referred to the document source although the item id is never reassigned. They are no longer counted among the set of total items.

* "CA" designates a former item for which consensus has been achieved in the judgment of the author, although not by any official process.

Items reaching this state are effected in the document source including the deletion of annotations and the elimination of obsoleted previous treatments.

Items in this state are still counted among the total of item but are no longer considered pending

* "CV" designates a former item for which consensus has been achieved and formally verified.

  Items in this state are not counted among the item totals. They may be kept in the table but only to indicate that the item id is still reserved.

* "DR" designates a former item which has been dropped, because it appears that working group acceptance of it, even with some modification, is unlikely.

  Such items no longer are referred to the document source although the item id is never reassigned. They are no longer counted among the set of total items.

When asterisk is appended to a state of "NM", "BC" or "BE" it that there has been adequate working group discussion leading one to reasonably expect it will be adopted, without major change, in a subsequent document revision.

Such general acceptance is not equivalent to a formal working group consensus and it not expected to result in major changes to the draft document,

On the other hand, once there is a working group consensus with regard to a particular issue, the document will be modified to remove associated annotations, with the previously conditional text appearing just as other document text does. The issue will remain in this table as a non-pending item. It will be mentioned in Appendix A of [I-D.dnoveck-nfsv4-security], to summarize the changes that have been made.

It is to be expected that these designations will change as discussion proceeds and new document versions are published. It is hoped that most such shifts will be upward in the above list or result in the deletion of a pending item, by reaching a consensus to accept or reject it. This would enable, once all items are dealt with, an eventual request for publication as an RFC, with this appendix having been deleted.

The consensus items in the following table can be divided into three groups, based on the associated numeric id.

* Those with ids less than 62 were created as part of the security document and transferred to this one as part of the document split.

* Those with ids between 62 and 65 are the result of splitting items created as part of the security document that now address issues in both documents

* Those with id 100 and above were created after the document split. In most case, there is no connection to material within the security document.

#	Type	...References...	Substance
3	BE	#3a in S 8.3	Conversion of mask bit descriptions from being about "permissions" to being about the action permitted, denied, or specified as being audited or generating alarms.
4	CI	#4a in S 2.3 #4c in S 8.3 #4d in S 8.3.2 #4e in S 8.3.5 #4f in S 8.3.6 #4g in S 8.4.1	Elimination of uses of SHOULD believed inappropriate in the descriptions of ACEs and clarification of ongoing use of SHOULD.
5	NE	#5a in S 8.3 #5b in S 8.3.5 #5c in S 8.3.6 #5d in S D.2.1	Changes needed in treatment of ACCESS, including the following: * ACCESS is listed as an operation in all cases in which one of the bits returned by the operation could be affected. There is information about possibly affected bits. * There is now a discussion of differences between the effect

			on authorization and that on other uses of the associated mask bits for ACEs not connected with authorization. Because ACCESS does not actually attempt access, it OK for use of of ACCESS not to be a recordable event, even though some servers, with an abundance of caution, might choose to record it.
7	BE	#7a in S 8.3.2 #7b in S 8.3.5	Clarification of relationship between READ_DATA and EXECUTE.
8	BC	#8a in S 8.3.2 #8b in S 8.3.5 #8c in S 8.3.6	Revised discussion of relationship between WRITE_DATA and APPEND_DATA.
9	CI	#9a in S 8.3.2 #9b in S 8.3.6	Clarification of how ADD_DIRECTORY relates to RENAME. We are assuming that the cross-directory and within-directory cases need to be treated differently.
10	WI	#10a in S 8.3.11	Possible revisions in handling of the masks WRITE_RETENTION and WRITE_RETENTION_HOLD.
11	BC	#11a in S 8.3.11	Explicit recommendation and requirements for mask granularity, replacing the previous treatment which gave the server license to ignore most of the previous section, placing clients in an unfortunate situation. Expect discussion but no objections.
12	BC	#12a in S 5.9 #12b in S 6.6	Revised treatment of directory entry deletion.

		#12c in S 8.3.20		
		#12d in S B.1		
		#12e in S 12.5		
13	BC	#13a in 8.4 #13b in 8.4.1 #13c in 8.4.2 #13d in 8.4.2.1	Clarify the use of ACE flags while attempting to put some reasonable limits on possible non-support (or variations in the support provided) for the ACE flags. This is to replace a situation in which the client has no real way to deal with the wide freedom granted to server implementations.
14	BC	#14a in S 5 #14b in S 5.8 #14c in S 7.1 #14d in S 7.2	Explicit discussion and use of the case in which aclsupport is not supported.
15	BC	#15a in S 5.8	Handling of the proper relationship between support for ALLOW and DENY ACEs.
16	NM	#16a in S 5.7	Discussion of coherence of acl, sacl, and dacl attributes.
26	CI	#26a in S 10.3 #26b in S 10.7.1 #26c in S 10.7.2 #26d in S 10.7.3 #26e in S 10.7.4	Decide how ACEs with who values other than OWNER@, Group, or EVERYONE@ are be dealt with when setting mode. Variant implementations exist. Clear rules/recommendations needed. Likely to be affected by new approach to draft-POSIX ACLs.
27	CI	#27a in S 10.3 #27b in S 10.5 #27c in S B.2	Concerns the possible existence of multiple methods of computing a mode from an acl that clients can depend on, and the proper relationship among these methods.

		#27d in S B.3	
28	BC	#28a in S 10.3 #28b in S 10.5 #28 in S 10.7.1	Decide how to address flags in mapping to/from reverse- slope modes.
29	BC	#29 in S 10.7.1	Address the coordination of mode and ACL-based attributes in a unified way for all minor versions.
30	CI	#30a in S B.4 #30b in S B.5 #30c in S 10.7.1 #30d in S 10.7.2 #30e in S 10.7.3 #30f in S 10.7.4	New proposed treatment of setting mode incorporating some consequences of anticipated decisions regarding other consensus items (#26, #28, #29)
31	RT	Gone.	Need to deal with mask bits ACE4_READ_ATTRIBUTES, ACE4_WRITE_RETENTION, ACE4_WRITE_RETENTION_HOLD, ACE4_READ_ACL to reflect the semantics of the mode attribute. No longer relevant because of rewrite of Section 10.7 and its use of the ACL_Choice attribute.
50	BC	#50a in S 5.9 #50b in S 8.4.3 #50c in S 8.4.3.1 #50d in S 8.4.3.2	Revise handling of "special" who values, making it clear for which ones "special" is a euphemism for "semantics-challenged".
51	BC	#51a in S 8.4.3	Clarify the handling of the group bit for the special who values.
61	RT	#61a in S 5.6	Proposal to distinguish support for UNIX and NFSv4 ACLS, depending on

		#61b in S 5.8 #61c in S 5.11 #61d in S 7 #61e in S 7.2 #61f in S B.2 #61g in S B.3	the results returned by the ACL_Support attribute. Modified to be conditional on the absence of ACL_Choice attribute because of the need to effectively handle hybrids. Many previous instances of this item now include Item #105 as well, since ACL_Choice, when supported, replaces the attempt to infer the semantic model based on material available in earlier minor versions. Now superseded by facilities associated with the ACL_Choice attribute.
62	BC	#62a in S 6.3 #62b in S 8.3.3	New/revised description of the role of the "sticky bit" for directories, with respect to ACL/ ACE handling. Needs to be considered together with Item #6 in the security document proper.
63	CI	#63a in S 10.1 #63b in S 10.3 #63d in S 10.8 #63e in S 10.10 #63f in S 10.11	Revised description of co-ordination of acl and mode attributes to apply to NFSv4 as a whole. While this includes many aspects of the shift to be more specific about the co-ordination requirements including addressing apparently unmotivated uses of the terms "SHOULD" and "SHOULD NOT", it excludes some arguably related matters dealt with as Consensus Items #26 and #27. Needs to be considered together with Item #25 in the security document proper.
64	WI	#64a in S 6.1	Discussion of issues related to the scope of the UNIX ACL model and the

			provisions made to handle it. Needs to be considered together with Item #56 in the security document proper.
65	RT	#65a 5.7 #65b 5.10 #65c 5.11	Designation of the Acl, Dacl, and Sacl attributes as Experimental in previous specifications even though still formally OPTIONAL. Note that this is separate from the possibility of sufficiently clarifying the description of the acl, dacl, and sacl attributes to make the Experimental designation unnecessary, or providing other means of semantic model discovery, which will be covered as Item #110. Needs to be considered together with Item #58 in the security document proper. Given recent developments including the lack of need, within this feature, for draft-POSIX ACLs support and definition of ACL_Choice.
101	BC	#101a in S 8.3.5	Inclusion of the action of READLINK as authorized by ACE4_READ_DATA
102	BC	#102a in S 8.3 #102b in S 8.3.10	Mask bits have to be dealt with that are not simply finer-grained correlates of one of the three POSIX privilege bits or of ownership.
103	NM	#103a in S 8.3 #103b in S 8.3.6 #103c in S 8.3.8	Classification of masks bits based on relationship to permission bits and existence of implementations. Discussion of detail expected but without this, the list was hard to understand.
104	BC	#104a in S 1	Presentation of UNIX ACLs as the

				basis of the feature, rather than
			#104b in S 1.2	the Windows-based NFSv4 ACLs, that
				contain features needed by a
			#104c in S 3	limited set of clients.
			#104d in S 3.2	
			#104e in S 3.3	
			#104d in S 3.1	
			#104d in S 3.1	
			#104g in S 6	
			#104h in S 6.4	
			#104i in S 6.1	
105	BC		#105a in S 1	Support for discovery of ACL
				extensions using the ACL_Choice
			#105b in S 1.2	attribute or by using inference
				rules, to help in those cases in
			#105c in S 3	which it is not supported.
			#105d in S 3.2	Includes necessary restrictions on
				server semantics to enable useful
			#105f in S 5.9	support to be provide when
				ACL_Choice is not supported (e.g.,
			#105g in S 5.11	in NFSv4.0)
			#105h in S 6	Presumes Item #104 has been
				implemented as well.
			#105i in S 6.1	
			#105j in S 6.4	
			#105k in S 6.5	
			#105l in S 6.6	
			#105m in S 7	
			#105n in S 7.1	
			#105o in S 7.2	
			#105p in S 8.1.2	

		#105q in S 8.1.3		
		#105r in S 8.4.2		
		#105s in S 8.4.3.1		
		#105t in S 10.5		
		#105u in S 10.7.1		
		#105v in S 12.5		
106	BC	#106a in S 8.3.5 #106b in S 8.3.6	More detail about cases in which OPEN is affected by ACE mask bits, including the dependence on the type of OPEN.	
107	BC	#107a in S 8.3.5 #107b in S 8.3.6	More detail about use of ACE4_WRITE_DATA and the dependence on the support for finer-grained bits in descriptions of ACE mask bits.	
108	BC	#108a in S 8.3.5	Distinguish mask bit treatments depending on the type of the objects	
109	BC		More detail about cases in which RENAME is affected by ACE mask bits including the dependence on the directories for which the mask bits, distinguishing the within-directory and cross-directory cases, and dealing appropriately with the rename-over case.	
110	BC	#110a in S 5.8 #110b in S 5.11 #110c in S 7 #110d in S 7.1 #110f in S 10.5	Make explicit reference to the ACL semantics provided by the server, assuming this can be known somehow, rather than by hand-wavily assuming that clients will somehow get by. Assumes that Item #61 or #105 is present or some replacement.	

		#110g in S B.3	
111	BC	#111a in S 8.3 #111b in S 8.3.5 #111c in S 8.3.6 #111d in S 8.3.10	Needs to be considered together with Item #66 in the security document which deal with parallel issues regarding POSIX-based authorization. Addresses more substantively the handling of the mask bits ACE4_{READ,WRITE}_NAMED_ATTRIBUTES.
112	BC	#112a in S 8.3.11	Address the validity/utility of ACE4_READ_ATTRIBUTES. This might be unnecessary, if ACL_Choice were implemented, since non-support would be a available as an option likely to be commonly chosen.
113	CA	None	Clarify how ACE mask bits defined in Section 8.3.6 are to be dealt with by clients when the server does not support those mask bits, as might be the case when the server supports the UNIX ACL model. No references any more. Assume issues dealt with elsewhere.
114	BC	#114a in S 1.2	Clarify the goals of the document as part of the rfc8881bis effort, given that we might be unable to immediately undo the damage created by the earlier approach and subsequent decades in which the consequences of the profound underspecification of the authorization semantics was ignored.
115	RT		Create a separate bit, if necessary, to govern the authorization approach used in draft-POSIX ACLs, in which an ACE is not allowed to partially satisfy an authorization request. No longer necessary with change in

			approach to draft-POSIX ACLs.
116	NM	#116a in S 6.3	Create behavioral restrictions in the form of residual SHOULDs, to provide ensure proper support for clients when ACL_Choice is not available. The focus will be on UNIX-oriented clients although better support for other clients should be considered. There could be disputes about how directive to be in tis context.
117	CA	#117a in Abstr. #117b in S 1 #117c in S 1.2 #117d in S 1.3 #117e in S 3.2 #117f in S 3.6 #117h in S 6.1 #117i in S 6.2 #117j in S 8.4.3.1 #117k in S 12.5	Clearly distinguish the draft-POSIX ACL models and the subset of the Windows ACL model that is compatible with a POSIX-based approach to authorization, referred to in this document as the "UNIX ACL model". In this document, we need to distinguish clearly those elements of the draft-POS/IX ACLs that are allowed behavioral variants within the NFSv4 ACL model and those that cannot be addressed that way within the context of this respecification effort. Given the group’s inability to extend the Windows ACL model to support draft-POSIX ACLs and the group’s adoptions of Rick’s POSIX ACL extension document, it makes sense to consider this issue effectively resolved as it has essentially decided that the assumption that the NFSv4 ACL model effectively superseded the draft POSIX ACL model is now understood to have been mistaken.
118	NM	#118a in S 5.2 #118b in S 5.7 #118c in S 5.10	Clarify issues related to proper denotation of the absence of ACLs when interrogating or setting ACL-related attributes. The current proposal is to use

		#118d in S 5.11	empty ace arrays. It is possible that this approach will result in controversy. Regardless of that choice, this issue needs to be addressed somehow.
119	BC	#119a in Abstr.	Needs to clearly state the requirements for support of Windows ACL semantics and how they relate to NFSv4's embrace of POSIX.
120	BC	#120a in S 5.3 #120b in S 5.5 #120c in S 12.4 #120d in A D.2 #120e in A D.2.2 #120f in A F.4	Provides additional detail and associated defect correction and extensions to properly address issues relating to ACL size limits.
121	BC	#121a in S 1 #121b in S 5.9 #121c in S 6.3 #121d in S 8.3 #121e in S 8.3.6 #121f in S 8.3.8	Support for ACE4_{READ,WRITE}_EXT_ATTRS.
122	BC	#122a in S 1 #122b in S 5.9 #122c in S 8.3 #122d in S 8.3.10	Support for ACE4_SACL_ACCESS.
123	BC	#123a in S 1 #123b in S 5.9	Support for CREATOR_{OWNER,GROUP}@

			#123c in S 8.4.1	
			#123d in S 5.9.2	
			#123e in S 8.4.3	
			#123f in S 8.4.3.1	
124	BC	#124a in S 1	Support for OWNER_RIGHTS.	
		#124b in S 5.9		
		#124c in S 5.9.2		
		#124d in S 8.4.3		
		#124e in S 8.4.3.1		
125	BC	#125a in S 8.4.1	Cleanup of INHERITED_ACE	
		#125b in S 10.7.1		
126	BC	#126a in S 8.3	Revision of treatment of ACE mask bit to focus on the arrays that provide the interface between server handling of various mask bits and set-mode/compute-mode logic later. Use of c++-oriented pseudocode to explain construction of these controlling arrays.	
		#126b in S 8.3.1		
		#126c in S 8.3.14		
		#126d in S 8.3.15		
		#126e in S 8.3.16		
		#126f in S 10.5		
		#126g in S 10.7.1		
127	BC	Re-organization of discussion of inheritance to treat server-effected propagation and client-managed propagation as alternatives,	#127a in S 5.10	
			#127b in S 5.11	
			#127c in S 8.2	
			#127d in S 8.2.1	
			#127e in S 8.2.2	

			replacing the previous diachronic presentation which treated the client-managed as an additional feature, which is how it was previously conceived.	#127f in S 8.2.3 #127g in A E.2

Table 7

The following table summarizes the issues in each particular state.

Type	Cnt	Detail
BC	32	8, 11, 12, 13, 14, 15, 28, 29, 50, 51, 61, 101, 102, 104, 105 106, 107, 108, 109, 110 111, 112, 114, 119, 120 121, 122, 123, 124, 125 126, 127
BE	2	3, 7
CI	6	4, 9, 26, 27, 30 63
NE	1	5
NM	4	16, 103, 116 118
NS	0	
WI	2	10, 64
Non-terminal	47	BC, BE, CI, NE NM, NS, WS
RT	4	31, 61, 65, 115
CA	2	113, 117
Terminal	6	RT, CA
All	46	Grand total for above table.

Table 8

C.2. Issue Changes

C.2.1. Issue Changes Until Acls-01

When the acls document was split from the overall security document the Consensus items related to ACLs were moved over, with their original item number from that document to this. In addition, five items that affected text in both documents were split and had new item numbers assigned for the acl-related portion of the item.

In addition, as a result of work to make server semantics more visible to the client, a substantial number of new issues, with item numbers over one hundred were added.

As a result, there were forty unresolved Consensus items in acls-01.

C.2.2. Issue Changes In Acls-02

As a result of a review of existing consensus items the following changes were made:

* Items #100 and #111 were merged.

* A new Item #114 was created. It deals with document goals.

* A new Item #115 was created. It deals with the partial-authorization-prohibition in the semantics of draft-POSIX ACLs.

* A new Item #116 was created. It deals with possible server behavior restrictions to ensure proper support to clients.

* A new Item #117 was created. It deals with better handling of the draft-POSIX ACL model

As a result, there are now forty-two unresolved Consensus items in acls-02.

C.2.3. Issue Changes In Acls-05

As a result of changes made in acl-05:

* The substance of item #117 was changed as it became clear that the issues previously associated with it could not be addressed as part of the respecification effort.

  Nevertheless, the new version of the item addresses the same gaps in the original specifications and is suitably treated as a replacement.

* A new item, #118, was added to deal with the representation of the absence of an acl as a value of an ACL-related attribute.

As a result, there are now forty-three unresolved Consensus items in acls-05.

C.2.4. Issue Changes In Acls-update-01

This section summarizes issue changes implemented in the final drafts of the individual draft (i.e., in -05 thru -07) and the initial drafts of the Working Group document (i.e., -00 and -01).

These changes are best understood as consequences of the current ongoing process leading to the decisions resulting in the solidification of the approach to the ACL features now present in the document proper. For a discussion of this evolution, see Appendix A.1

These changes of approach have resulted in a large set of changes to the enumerated consensus issues together with corresponding plans for their eventual resolution.

* A large number of issues are now classified as "Believed Correct", since the author believes that new approach has eliminated the possibility of alternate approach.

* A number of issues have been moved to a terminal state. These includes some moved to RT because they are no longer necessary in the current framework and a number moved to CA based on the author’s expectation that, in the new framework, it s only a matter of time for the issue to be put behind us as the document continues its development.

* As described in Appendix C.3, there has been a classification of pending issues to guide their discussion, and track the result of that discussion as the document moves forward.

The following individual changes were made:

* Issues converted to BC including the following:

  Issues 8 and 11 moved from CI.

  Issues 101, 110, and 114 moved from NM.

  Issues 62, 102, 104, 105 moved from NE.

  Issues 111 and 112 moved WI.

Issue 119 was created in BC.

* Issues converted to the terminal state RT:

  Issue 65 moved from NS.

  Issue 31 moved from WI.

  Issue 115 moved from CI.

* Issues converted to the terminal state CA:

  Issue 113 moved from NE

  Issue 117 moved from NM

C.2.5. Issue Changes In Acls-update-02

As part of a review of issued marked Delay-Likely, it was discovered that issue #28, although previously dealt with in the drafting of text to satisfy Consensus Item #30 had been left in an anomalous state:

* The Iten was left marked WI, even though appropriate text had been written as part of dealing with Item #30.

* Despite the expected adequacy of new text, the item was marked Delay-Likely.

  Although the text satisfying Item #30 could address the substance of Issue #28 as well, it made sense to keep it on the list for later discussion.

In addition, the following changes were made:

* The Issue #28 was moved from state WI to state BC.

* Issue 120 was created in state BC.

C.2.6. Issue Changes In Acls-update-03

Issues 121, 122, 123, 124, 125, 126,and 127 were created in state BC.

C.3. Issue Priorities

As a result of the history of this document, there are two important classes of issues that should be dealt with expeditiously to allow us to make progress on this document and complete the v4.1 respecification effort, despite the plethora of problems in the specification of ACLs in exiting standard-track documents.

* There a large set of issues whose discussion is now long overdue, as a result of lack of working group discussion of the issues raised by the existing approach to specification of this feature.

  Ten of these issues listed in Table 9 should be non-controversial, making it desirable we get these out of the way in order to focus on more difficult matters.

* Another set of important issues are those of more recent origin and reflect the important need, previously unaddressed, to recognize important behavioral choices as OPTIONAL features and make the status of these choices accessible to clients.

  Eight of these issues listed in Table 10 can to be prioritized so that the working group has a clearly understood shared understanding of the general approach being taken towards ACLs.

Each of these tables list the issues in numerical order with columns for a brief Issue description and a Status chosen from among the possibilities below:

Terminal (T) Issue already in a terminal state.

  Need to focus on:

  * Deleting the tracing of the consensus item as no longer necessary once consensus has been achieved.

  * Updating text to no longer be conditional on the deleted consensus item. This will be necessary to simplify the document for review outside the Working Group as it moves forward.

Near Terminal (NT) Issue likely close to a terminal state but needs review to ensure airing of possible differences of opinion.

  Need to focus on:

  * Providing occasions for discussion of potential issues using on-list discussion and available time at interims.

* Deciding when a terminal state is appropriate.

* Effecting the transition to a terminal state and potentially following up as one would for a terminal state, as described above.

Delay Likely (DL) Issue for which disagreement can be anticipating, making it necessary that we focus initially on clarifying the nature of the disagreement as a prelude to eventually resolving it.

Need to focus on:

* Clarifying the nature of the disagreements and transforming them into disagreements about proposed text, rather than a general difference about approaches to writing documents or feelings about ACLs.

* Understanding where sets of issues are related so that they can be dealt with together, to avoid inconsistencies of approach in the resulting document.

  Since some issues will need to be coordinated, in the same way, with consensus issues dealt with in other documents (e.g., security), these need to be dealt with similarly

* Providing occasions for discussion of potential issues using extensive on-list discussion to make it likely that participants are informed of the issues so a decision can be arrived at.

Resolution of the following issues is long overdue, most likely because of lack of discussion of the underlying document. The document's recent adoption by the Working Group needs to be followed up on, to make sure there is adequate review of the document that is eventually published.

Item	St.	Discussion
4	DL	Resolving cases in which SHOULD is used inappropriately. Most or all of these are cases in which the definition in [RFC2119] is not adhered to. Might be some controversy about the replacement in specific cases, but we need

		to isolate any disagreements so that we can focus on resolving them.
7	NT	Clarification of READ vs. Execute. Believe this is editorial and expect the working group to agree.
8	NT	Clarification of Write vs, Append The previous concern about possible compatible issues is no longer relevant since there is an ACL_Choice flag for this, so that all previously valid approaches remain valid.
14	NT	Deals with the case of ACL_Support not being supported. Fixing this gap should be uncontroversial.
15	NT	Fix relationship of support for ALLOW and SENY. What is there is clearly bogus. Can be sure my replacement is better but there may people who want something else.
16	NT	Coherence requirements for acl, sacl, dacl. Although this is new material, it is unlikely to generate controversy. I think the new text merely clarifies existing expectations
27	DL	Multiple methods of computing mode. There is likely to be controversy about details but it seems that a least two methods will be required, especially since the protocol has been that way from the beginning.
29	NT	Making sure that the coordination requirements for modes and ACLs apply to all minor versions. There is some text that suggests otherwise

		but if the same requirements do not hold, it is hard to see what can be said or how clients could be asked to deal with the resulting vacuum.
30	DL	New proposed treatment of setting mode incorporating some consequences of anticipated decisions regarding other consensus items (#26, #28, #29) There might be objections and this could change but the existing confusion cannot continue as it has.
50	NT	Handling of special "who" values. Expect a big yawn and difficulty getting comments but controversy is unlikely.
51	NT	Clarify the handling of the group bit for the special who values. Needs to be resolved somehow.
101	NT	Inclusion of the action of READLINK as authorized by ACE4_READ_DATA Can’t imagine any controversy about this.

Table 9: Issues For Which Resolution is Long Overdue

Resolution of the following issues needs to be done quickly because they are connected to important elements of the fundamental approach o description of the ACL changes that is now anticipated. As a result, if there is disagreement on these issues, it is best to clarify and resolve them relatively quickly.

Item	St.	Discussion
11	NT	Explicit recommendation and requirements for mask granularity, replacing the previous treatment which gave the server license to ignore most of the previous section, placing clients in an unfortunate situation. Expect discussion but no objections.

64	DL	Clarification of the role, scope, and handling of the UNIX ACL model. It is important to be clear about this because the existing specs, while making great efforts to allow use of this model, are very unclear about the scope of that model or the range of behaviors to be supported. The decision of make this model the required core of supported has raised the importance of this issue. Expect issues about detail but do not expect disagreement on the need to do this.
102	NT	Handling of orphan mask bits. Discussion may turn up some useful suggestions. In any case, the working group has to make decisions about each of these as was not done when they entered the protocol in RFC3010.
104	NT	UNIX ACLs as basic. Might be controversial but the issue needs to be discussed regardless.
105	NT	ACL_Choice attribute. Expect a lot of controversy, but the inability of clients to determine the presence of extensions needs to be addressed somehow.
109	NT	More detail about cases in which RENAME is affected by ACE mask bits including the dependence on the directories for which the mask bits, distinguishing the within-directory and cross-directory cases, and dealing appropriately with the rename-over case. Can’t see what would be controversial here. Nevertheless, working group feedback is expected to be helpful.

110	NT	Make explicit reference to the ACL semantics provided by the server, assuming this can be known somehow, rather than by hand-wavily assuming that clients will somehow get by. Could be a shock for some but the issue needs to be clearly understood and a decision made.
113	T	Clarify how ACE mask bits defined in Section 8.3.6 are to be dealt with by clients when the server does not support those mask bits, as might be the case when he server supports the UNIX ACL model. Don’t expect much controversy on this.
114	NT	Clarify the goals of the document as part of the rfc8881bis effort, given that we might be unable to undo the damage created by the earlier approach taken and subsequent decades of neglect. Expect controversy which is precisely why we need to have this discussion instead of putting it off repeatedly.
117	T	Make changes to better describe the needs relating to the support of draft-POSIX ACLs. The scope of these changes has changed recently do that the needs cannot be addressed as part of the respecification effort. Expect this discussion to focus other ways of addressing draft-POSIX ACL support in NFv4.2 extensions, as discussed in Appendix E. Those matters are not part of this consensus item, however.
118	DL	Make changes to clearly describe how the absence of an ACL is represented when interrogating an ACL-related attribute or when setting one in order to delete an ACL. There might be controversy about the proper form but we need to be clear that there is

		such a form, particularly to allow the future support of multiple ACL models.
119	NT	Needs to clearly state the requirements for support of Windows ACL semantics and how they relate to the continuing association of NFSv4 with POSIX semantics. May be extensive discussion but cannot see significant disagreement.
120	NT	Work to better handle filesystem limits on ACE size and discuss the possibility of transport-imposed limits on ACL sizes. Needs a careful review of the details of the additions.

Table 10: Issues that Need to be Addressed Soon to Reflect Current Direction

The table below lists issues that are outside the two special groups listed above. Despite the higher priority of issues in the remaining groups, it makes sense to make future priority decisions based on the status assigned, rather that the specific list it was placed on. In that context:

* Near-Terminal issues from this still need to be followed up to make the list of active consensus issues manageable.

* Delay-Likely issue still need the necessary analysis and might be related to similar issues dal with in the earlier lists.

Item	St.	Discussion
3	NT	Conversion of mask bit descriptions from being about "permissions" to being about the action permitted, denied, or specified as being audited or generating alarms. Arguably non-editorial but I don’t see many grounds for dispute.
5	DL	Changes needed in treatment of ACCESS.
9	DL	Clarification of how ADD_DIRECTORY relates to

		RENAME. We are assuming that the cross-directory and within-directory cases need to be treated differently.
10	DL	Possible revisions in handling of the masks WRITE_RETENTION and WRITE_RETENTION_HOLD. Not sue this is right need WF discussion.
12	NT	Revised treatment of directory entry deletion. Don’t expect major issues.
13	NT	Attempt to put some reasonable limits on possible non-support (or variations in the support provided) for the ACE flags. This is to replace a situation in which the client has no real way to deal with the freedom granted to server implementations. Don’t expect significant controversy.
26	DL	Decide how ACEs with who values other than OWNER@, Group, or EVERYONE@ are be dealt with when setting mode. Variant implementations exist. Clear rules/ recommendations needed. Likely to be affected by new approach to draft-POSIX ACLs.
28	DL	Decide how to address flags in mapping to/from reverse- slope modes.
31	T	Need to deal with mask bits ACE4_READ_ATTRIBUTES, ACE4_WRITE_RETENTION, ACE4_WRITE_RETENTION_HOLD, ACE4_READ_ACL to reflect the semantics of the mode attribute. No longer relevant because of rewrite of Section 10.7 and its use of the ACL_Choice attribute.
61	T	Proposal to distinguish support for UNIX and NFSv4 ACLS, depending on the results returned by the ACL_Support attribute.

		Modified to be conditional on the absence of ACL_Choice attribute because of the need to effectively handle hybrids. Many previous instances of this item now include Item #105 as well, since ACL_Choice, when supported, replaces the attempt to infer the semantic model based on material available in earlier minor versions. Now superseded by facilities associated with ACL_Choice attribute.
62	DL	New/revised description of the role of the "sticky bit" for directories, with respect to ACL/ACE handling. Needs to be considered together with Item #6 in the security document proper, which may lead to delay.
63	DL	Revised description of coordination of acl and mode attributes to apply to NFSv4 as a whole. While this includes many aspects of the shift to be more specific about the coordination requirements including addressing apparently unmotivated uses of the terms "SHOULD" and "SHOULD NOT", it excludes some arguably related matters dealt with as Consensus Items #26 and #27. Needs to be considered together with Item #25 in the security document proper.
65	T	Designation of the Acl, Dacl, and Sacl attributes as Experimental in previous specifications even though still formally OPTIONAL. Note that this is separate from the possibility of sufficiently clarifying the description of the acl, dacl, and sacl attributes to make the Experimental designation unnecessary, or providing other means of semantic model discovery, which will be covered as Item #110. Needs to be considered together with Item #58 in the security document proper.

		Given recent developments including the lack of need, within this feature, of draft-POSIX ACLs support and definition of ACL_Choice.
103	NT	Classification of masks bits based on relationship to permission bits and existence of implementations. Discussion of detail expected but without this, the list was hard to understand.
106	NT	More detail about cases in which OPEN is affected by ACE mask bits, including the dependence on the type of OPEN. No major issues expected.
107	NT	More detail about use of ACE4_WRITE_DATA and the dependence on the support for finer-grained bits in descriptions of ACE mask bits. No issues expected.
108	NT	Distinguish mask bit treatments depending on the type of the objects No controversy expected.
115	T	Create a separate bit, if necessary, to govern the authorization approach used in draft-POSIX ACLs, in which an ACE is not allowed to partially satisfy an authorization request. No longer necessary with change in approach to draft-POSIX ACLs.
116	DL	Create behavioral restrictions in the form of residual SHOULDs, to provide ensure proper support for clients when ACL_Choice is not available. The focus will be on UNIX-oriented clients although better support for other clients should be considered. There could be disputes about how directive to be in tis context.

Table 11: Othe Issues to be Addressed.

The following table summarizes the results of the analysis of the three issue lists above. Important conclusions are:

* The long-overdue issue are mostly Near-Terminal and should be moved to a terminal state within the next few drafts.

  The two Delay-Likely issues need some follow-up as well

* The direction-setting issues are also mostly Near-Terminal and should be moved to terminal expeditiously as well.

  The two Delay-Likely issues need some follow-up as well

* The remaining issue contain many Delay-Likely issues that need work along with the Delay-Likely issues form the other lists.

  Thee seven Near-Terminal Issues need to be addressed together with those from the other lists

* Overall, we seem to have moved from having forty-five unresolved consensus item to having thirty-nine of which twenty-five are Near-Terminal.

  When those are resolved, there will be fourteen Delay-Likely issues to deal with. If we follow up on those appropriately, they should not pose an insurmountable barrier to getting this document done.

Source	Group	Term	Near	Delay	Pending
Table 9	Long Overdue	0	10	2	12
Table 10	New Direction	2	8	2	10
Table 11	Other	4	7	10	17
Totals		6	25	14	39

Table 12: Count of Consensus Issues

It is also useful to look at issues grouped by current status.

State	Term	Near	Delay	Pending>
BC	0	23	0	23
BE	0	2	0	2
CI	0	0	6	0
NE	0	0	1	1
WI	0	0	3	3
RT	4	0	0	0
CA	2	0	0	0
Tot.	6	29	10	39>

Table 13: Consensus Issues by State

C.4. Handling of Consensus Issue for which Delay is Likely

The following table provides a proposed division of these issues into a set of groups.

Group		Issues		Sections	Discussion
Name	Ref	Name	Num		
Early	C.4.1	SHOULD	#4/L/CI	2.*, 7.*	Issues that need to be prioritized because they are early in the document.
		Absence	#118/D/NM	4.*	
WithSec	C.4.2	Sticky Bit	#62/O/BC	5*, 7.*	Issues whose resolution needs to be coordinated with related issues in the

					security document.
		Attr Co-ord	#63/O/CI	10.*	
		Unix ACL	#64/D/WI	5.*	
		Named Attr	#111/O/BC	7.*	
BitDesc	C.4.3	ACCESS	#5/O/NE	7.*	Issue whose resolution involves further clarification of the roles of the various ACE bits.
		ADD_DIRECTORY vs RENAME	#9/O/BC	7.*	
		Retention Bits	#10/O/WI	7.*	
		READ_ATTRIBUTES	#112/O/BC	7.*	
Combine	C.4.4	Comb Misc Who	#26/O/CI	10.*	issue whose resolution involves the coordination of authorization attributes.
		Compute Mode	#27/L/CI	10.*, F.*	
		Flags Reverse	#28/O/BC	10.*, F.*	
		Rewrite Set Mode	#30/L/CI	10.*, F.*	

Table 14

C.4.1. Delayed Issues That Appear Early in the Document

The issues listed below form a set that are used in the early sections of the ACL document. It makes sense to get these done soon so that once these are addressed, the basics of the document are no longer conditional on consensus items yet to be addressed.

* Item #4 addresses establishing, for the document as a whole a correct pattern of use for the BCP14-defined term SHOULD. This is necessary because of a confusing pattern of use of this term in earlier specifications.

  This issue involves potential compatibility issues and is currently in state CI, with the anticipation of likely delay recorded in the long-overdue list.

  Although there may be disagreements because of the lackadaisical way this term has been treated in the past, we have a good way to address potential incompatibility issues relying on a correct use of SHOULD with a clear indication that accommodating earlier specs is the only valid reason to bypass the recommendation. In addition, ACL_Choice allows the client information about server characteristics/compliance.

* Item #118 addresses issues regarding indicating the absence of an acl, dacl, or sacl value for a file system object.

  This issue is in state NM, with the anticipation of likely delay recorded in the direction list.

  There are a number of possibilities and disagreement is likely but the WG should be able to pick one.

Once these issues and some near-terminal one are resolved and the resolutions are confirmed, the document will not have Consensus issues in the first four top-level sections.

C.4.2. Delayed Issues That are to be Addressed Together with Security Document

This issues are better addressed together since they all need to be coordinated with related issues in the security document. For this reason adoption of the security document is an important preparatory step that needs to be attended to promptly.

* Item #62 deals with the semantics of the "sticky" bit for directories, which had not been discussed in previous specifications, even though it has an important role the authorization semantics of delete.

  This issue needs to be addressed together with Item #6 for the security document.

  This issue is in state BC, with the anticipation of likely delay recorded in the "Other" list.

  Discussion and review of details is necessary and expected. There may be some unwillingness to recognize that this has been left undone so long because of the groups lack of concern with authorization, but tis need to be addressed in the security document, while the discussion in this one has to be compatible although it is expected to be more complicated.

* Item #63 deals with the general subject of attribute coordination between POSIX authorization and that controlled by NFSv4 ACL.

  This issue needs to be addressed together with Item #25 for the security document.

  This issue is in state CI, with the anticipation of likely delay recorded in the "Other" list.

  Matters related to this coordination were not previously addressed, most likely because of a lack of interest in authorization issues. While there is room for disagreement on how these issues are best addressed, it is hard to imagine disagreement about the fact that coordination is necessary.

* Item #64 deals with the UNIX ACL model ad its role within the NFSv4 ACL model

  This issue needs to be addressed together with Item #56 for the security document.

  This issue is in state WI, with the anticipation of likely delay recorded in the "Direction" list. The expected working group input is expected to be about preferred terminology, rather than the need to clearly explain this pattern of use.

  Given the direction of the working group’s approach to draft-POSIX ACLs, I don’t expect anyone with having issues with the need to distinguishing draft-POSIX ACLs and the subset compatible with the NFsv4 ACL model.

* Item #111 deal with authorization related to named attributes

  This issue needs to be addressed together with Item #66 for the security document.

  This issue is in state BC, with the anticipation of likely delay recorded in the "Other" list.

  Do not expect much disagreement about this.

Once the issues in this and the previous section and some near-terminal ones are resolved and the resolutions are confirmed, the document will not have Consensus issues in the first five top-level sections.

This will leave the following Consensus Item groups to be addressed:

* The issues in Appendix C.4.3 deal with Consensus issues referenced in top-level Section 7.

* The issues in Appendix C.4.4 deal with Consensus issues referenced in top-level Section 10 and Appendix F.

C.4.3. Delayed Issues That Involve ACE Bit Mask Specification

This section deals with Consensus Issues that arose in connection with the need for correct and complete description of the actions covered by the ACE mask bits, that have not been dealt it in other groups.

The need for extensive work in this area derives from a number of gaps in previous specifications. In addition to a general lack of attention to the details of authorization, an important factor is the mistaken belief, relied upon in previous specifications, the differences among various implementations with regard to permission granularity can be dealt with simply by allowing servers to make their own choices, without work on the part of the specification.

This leaves us with the following Consensus Issues to address in this area.

* Item #5 deals with clarifying how the ACESS operation is dealt with by the various ACE mask bits,

  This issue is in state NE, with the anticipation of likely delay recorded in the "Other" list.

Expect extensive discussion of this issue but don’t expect controversy about making the handling more explicit. There may be multiple opinions about how to deal with ACCESS for AUDIT and ALARM ACEs but believe these will be resolvable.

* Item #9 deals with the interaction of ACE4_READ_DIRCTORY and RENAME. This was dealt with confusingly in previous specifications whose structure was not conducive to dealing separately with within-directory and cross-directory cases of RENAME

  This issue is in state CI, with the anticipation of likely delay recorded in the "Other" list.

  Want the discussion to focus on potential compatibility issues. If there are any, may need to address specially in ACL_Choice.

* Item #10, deals with ACE mask bits that deal with retention attributes.

  This issue is in state WI, with the anticipation of likely delay recorded in the "Other" list.

  The likely discussion will need participation of someone who is familiar with an implementation, to provide expert-level input.

* Item #112 deals with ACE4_READ_ATTRIBUTES, whose denial prevents GETATTR on the current file object. Given the lack of the possibility of such denial in POSIX file systems, anticipate many clients not be expecting such denial.

  This issue is in state BC, with the anticipation of likely delay recorded in the "Other" list.

  With the inclusion of ACL_Choice, Windows-oriented clients that need this can heck for server support while POSIX clients can be warned of support for this. The issue, about which I expect controversy, is how exactly to characterize this exceedingly non-POSIX semantics and wat advice to give, if any, about how to deal with it.

C.4.4. Delayed Issues That Involve Attribute Coordination

This section deals with Consensus Issues that arose in connection with the revision of text dealing with the interactions of multiple authorization-related attributes.

The need for extensive work in this area derives from a number of gaps in previous specifications. In addition to a general lack of attention to the details of authorization, an important factor is the extraordinary level of deference accorded to server preferences in this area and the corresponding unwillingness to suppress such differences, in order to enable interoperability.

The creation of ACL_Choice provides a way out of tis situation but it needs to be supported by choices as to the correct way to deal with these issues, leaving many existing implementations with work to do to enable effective interoperation until some future minor version ceases to be so tolerant of some choices currently allowed.

This leaves us with the following Consensus Issues to address in this area.

* Item #26 deals with the expected/recommended fate of ACEs with a wide range of Who values (i.e., all except OWNER@, GROUP@, EVERYONE@) when setting mode. Although it seems that such ACEs need to be deleted, existing specifications do not say that and implementation that do other things appear to do exist.

  This issue is in state CI, with the anticipation of likely delay recorded in the "Other" list.

  Expect controversy, although it to be made more resolvable by the existence of ACL_Choice and the commitment not to disallow existing behavior post facto. In addition, the expected new path to draft-POSIX ACL support may help focus the group’s concerns on a providing a workable ACL future rather that trying to accommodate two very different ACL models in the same framework.

* Item #27 concerns the computing of a mode value corresponding to an ACL being set, and modifying the excessively laissez-faire approach to this matter manifested in previous specifications.

  This issue is in state CI, with the anticipation of likely delay recorded in the "Long Overdue" list.

  Anticipate long discussion, dealing with the issues that were ignored when the original text regarding this issue was written. The basic issue is that because of ACL’s more complicated structure, there a large set of ACLs that cannot be represented by an equivalent mode.

Given this fact, individual will come to the fore and will inevitably be different. The argument has to be made that server preferences are not to be consulted here and that the clients need to rely on the spec to define how servers are to deal with this matter.

Although I expect considerable reluctance by some, I think we will have to accept the idea that defining a "right" way to do this, is part of transforming NFS to be standards-based protocol, even though NFSv3 was an implementation-based protocol that was successful for a considerable time.

* Item #30 deals with the changes to make to existing ACLs when the mode attribute is set. A decision was made to write a new description of this area, which needs to be discussed and reviewed. An important part of that discussion should cover potential compatibility issues, even though it is expected that the presence of ACL_Choice will ameliorate any issues.

  This issue is in state Ci, with the anticipation of likely delay recorded in the "Long Overdue" list.

  Expect extensive discussion about cases that might be difficult to deal with such as the handling of inherit ACEs that are not inherit-only.

* Item #28 deals with the correct handling of ACE flags when setting modes. Although previous specifications contained text that incorrectly ignored these, the new text written to satisfy item #30, should explain where the ACEs, rather than the flag bits are to be ignored.

  This issue is in state BC, with the anticipation of likely delay recorded in the "Other" list.

  Expect any necessary discussion to have already happen as part of the discussion of Item #30.

Appendix D. Prospective ACL_Choice Changes

D.1. Possible Simplifications

Simplification of following items is likely to be necessary:

* The handling of ACC4IN_ODDMB could be simplified once we had more information on the actual range of behavioral variation shown by existing server implementations.

* There are a number of ACE mask bits that look likely to be deleted at some point. These include ACE4_READ_ATTRBUTES and others for which no server implementation has been brought to the working group’s attention.

  To delete these it would be necessary to make an assessment of client need leading to a Consensus.

Work on these two areas is likely to be mutually reinforcing since the a reduction in the number of attributes to be reported within ODDMB will make it easier to classify the implementations of the rest. After all, in the likely event that some of these bits have only a signed server implementation, there is no possibility of behavioral variants.

D.2. Possible Additions

Excluding those connected with ACL_Choice simplification, we will discuss a number of possible additions to provide implementation information whether using the ACL_Choice attribute o similar extensions:

* A set of potential additions that need to be considered for possible future draft described in Appendix D.2.1. Some of these, might be deferred until subsequent minor versions

* [Consensus Needed (Item #120d)]: Possible extensions regarding the handling of ACL size limits described in Appendix D.2.2 These need to be considered for inclusion in later minor versions.

D.2.1. Possible Near-Term Additions

There are two likely sources of potential additions in that might be added to subsequent drafts and minor versions:

* Discovery of new behavioral variants in existing server implementations.

  While it could be argued that many of these are instance of flawed, non-compliant implementations, the general tenor of existing specifications may make it difficult to put existing implementations outside the set of acceptable choices.

  In view of the need to accommodate file system implementations with widely varying heritages, the most likely way of dealing with divergent implementations is to treat them as behavioral variants and deal with the groups advice/recommendations in Section 12.5.

* [Consensus Item #5d]: Report on cases in which a server treats the use of ACCESS as a recordable event.

* The potential need for finer authorization granularity.

  Although there is a need for removal of granularity overshoot, there may be need for additional granularity to deal with new functionality such as extended attributes

D.2.2. Possible Additions Related to ACL Size Limits

[Consensus Needed (Item #120e), Through end of section]: There are a number of likely implementation techniques that raise potential problems for the current approach to reporting ACL size limits:

(1): Optimized ACE encoding methods that cause the size taken up by various ACEs to vary, making it hard for limits expressed as ACE counts to allow for ACL with high ACE counts since the determination of actual length is not possible for the client.

Although various forms of ACE encoding may give rise to varying ACE space requirements, the most common source of such variability derives from filesystems which represent principals as variable-length identifiers.

(2): There are cases in which a limit on ACL size derives from ACLs being stored in a limited-size area shared with other features. As a result, limits on the size of the ACL will vary depending on the use of these other features.

A common instance of this sort arrangement arises when ACLs are stored together with miscellaneous attributes known as "extended attributes".

While there are ways that the above sources of ACE limit variability could be addressed by extensions to prevent the client receiving the errors introduced in Section 5.3, we will explain below why such extensions not address the problem, and that the focus needs to stat on giving clients reliable information as to the consequences of server space restrictions while leaving servers free to address implementation issues to provide ACLs of the size clients need.

There are two possible reporting extensions to consider:

(A): A per-file attribute with a structure similar to that of ACL_Choice could avoid ACL size errors caused by (1) but would leave (2) unaddressed

(B): A TestAcl operation could be used to avoid ACL size errors caused by both (1) and (2).

Despite the ability of such extensions to allow suppression of ACL size errors, there is good reason to question the value of such section as we think about how ACL are used to provide security for data stored within a filesystem. If the user needs ACLs of a significant size to effect security of the desired granularity, then getting an error to that effect of finding out that an error is likely are equally undesirable, since there is no way to provide the desired security if such size limits exist.

The ACL size limit information described in Section 12.4 is focused on providing, at an early stage, the limits the client will face as the security architecture and its granularity and ACL size requirements are formulating. Because low size limits restrict the chosen security architecture, clients might find a particular filesytems' ACL support uncongenial and use another one unless and until adequate support is provided with higher limits.

Despite the fact that, for many implementers, significant filesystem implementations are not available, implementers are not without recourse to deal with situations in which ACL size limits might make the NFSv4 ACL implementation unusable for a large set of clients:

* With regard to (1), the range of ACE sizes can generally be expected to be small so that the maximum ACE size is limited. As a result, if the space available for ACE is significant, large ACE counts can be provided to clients.

* With regard to (2), although it might be impossible to switch the filesystem away from the sharing of ACL space with data to support other features, it will generally be simpler to reserve space for significant-size ACL within the common pool. While this will limit the space available for other features and limit the size ACLs below what could be achieved using the full shared pool, they could often provide a substantial limit guarantee that many clients can use to advantage.

  In light of the above, filesystems could provide the ability to set up filesystems with varying levels of ACL space reservation.

Appendix E. Future Handling of Draft POSIX ACLs

Because of the large semantic differences between the NFSv4 and draft-POSIX ACL models, the use of mapping between these two models needs substantial supplementation in order to be effective. The following issues would need to be addressed:

(A) Support ACL inheritance using the default ACL model.

(B) Make it clear that, when both are supported, which authorization model a given ACL requires.

(C) Provide the ability to specify the mask which is often needed in many draft-POSIX ACLs.

Ways of dealing with these are discussed in Appendix E.1.

An additional issue that needs to be addressed concerns the lack of support for the automatic propagation of changes in inheritable ACEs downward in the naming hierarchy, Although this feature had its origin within Windows and has never been addressed in the draft-POSIX ACL model, it is worth investigating whether an analogous feature could be provided by an extended version of draft-POSIX ACLs, as explored in Appendix E.2/-

E.1. Ways of Addressing Existing Issues with Draft-POSIX ACLs

The following ways of accomplishing these ends have been considered:

* Defining new ACE types to address (B) and (C) while addressing (A) by defining a new default-dacl attribute.

  This would have the advantage of expanding the NFSv4 model to include the draft-POSIX model as a sub-model and enabling the default ACL approach to inheritance to be used more widely.

* Defining separate access-acl and default-acl attributes, oriented to the existing structure of draft-POSIX ACLs as described in [Gr端nbacher]. Such an approach is described in [I-D.ietf-nfsv4-posix-acls].

  This has the advantage of being more comprehensible and usable to those that have experience with draft-POSIX ACLs, whether locally or via a sideband protocol associated with NFSv3

E.2. Ways of Providing Equivalents for Windows "Automatic" ACL Inheritance

[Consensus needed (Item #127g), through end of section]: Although client-managed inheritance of draft-POSIX ACLs seems not to have been discussed previously, it is desirable for the same reason that "automatic" inheritance was implemented for Windows ACLs. As a result, we need to consider:

* Whether there is expected to be sufficient time for this feature to define it, most probably in the form of an NFSv4.2 extension.

  There is likely to be a considerable time before it makes sense to make this decision. This unlikely to be ready for decision soon after [I-D.ietf-nfsv4-posix-acls]. is published as an RFC. However, this RFC would need to provide the ability to eventually extend draft-POSIX to support this form of inheritance.

* It needs to be considered whether there might be troublesome interactions between the draft-POSIX ACL model and this form of inheritance. Our previous experience with this model has made it necessary that we carefully examine this issue before deciding that these can be addressed together.

  Rather unexpectedly, it turn out that the draft-POSIX approach to inheritance is easier to adapt to client-managed propagation of default ACL changes. This is primarily because default ACLs are inherited wholesale rather than in a finer-grained fashion in which individual ACEs are inherited, with the possibility of conflicts needing to be dealt with. An additional simplification is that the draft-POSIX ACLs are order-independent.

Despite the favorable expectations from the above analysis, it appears that this extension will require the addition of a flag word to the posix_default_acl attribute proposed in [I-D.ietf-nfsv4-posix-acls] in order to simplify the eventual upgrade path when this form of inheritance is defined as an extension. Until that point, the flag word will be available for future extension including the eventual definition of flags like those defined in Section 8.2.3/

Appendix F. Possible Future Extensions

There are a number of likely extensions that might need to be considered in a later minor version.

These include the possible Addition of ACE mask bits controlling the authorization of the interrogation and modification of extended attributes, as discussed in Appendix F.1.

F.1. Finer-grained Authorization Support for Extended Attributes

Currently, authorization for the modification or interrogation of extended attributes, is handled in a coarse-grained fashion:

* Authorization of the modification of these attributes is controlled by the write permission bit if there is no ACL.

When there is an ACL, authorization is controlled by a single ACE mask bit: ACE4_ADD_FILE in the case of directories and ACE_WRITE_DATA in the case of other types of objects.

* Authorization of the interrogation of these attributes is controlled by either read permission bit or the execute permission bit if there is no ACL.

  When there is an ACL, authorization is controlled by a the allowance associated with the or of two ACE mask bits.

  The first is ACE4_LIST_DIRECTORY in the case of directories and ACE_WRITE_DATA in the case of other types of objects.

  The second is ACE4_EXECUTE.

It needs to be considered whether it is worthwhile to provide finer-grained authorization in a fashion similar to the way in which reading and writing of named attributes is controlled, i.e., with separate ACE mask bits ACE4_READ_NAMED_ATTRIBUTES and ACE4_WRITE_NAMED_ATTRIBUTES.

It is worth noting that the current situation is inconsistent in that [RFC8276] defines extensions to ACCESS "to provide fine-grained access control to query or modify extended attributes", there are no fine-grained ways to control how these bits are set.

The following possibilities need to be considered:

* Creation of new mask bits ACE4_READ_XATTR and ACE4_WRITE_XATTR parallelling ACE4_READ_NAMED_ATTRIBUTES and ACE4_WRITE_NAMED_ATTRIBUTES with similar mappings when not supported.

* Creation of new mask bits ACE4_READ_XATTR, ACE4_WRITE_XATTR, and ACE4_LIST_XATTR matching the set of new ACCESS bits.

* Creation of new access bits that treat extended attributes together, along line similar to be sed in Windows.

F.2. Client Choice Regarding Mode Display

While the existing handling of this issue makes this a matter of server choice, it is really not clear why the server’s choices have any relevance here.

When a file has an ACL, it controls file authorization and the mode's role is limited to providing a value for display. Given this fact, it is unclear why the server's preferences in the matter have any real relevance and that, since this does not have any effect on what other clients see, the choice of what to do for display is not up to the requesting client.

If the client continues to interrogate the mode attribute there is no way to avoid the server's preferences in this becoming controlling, for no good reason.

A new read-only per-object attribute which we will call modeinfo could provide separate information regarding the permissions of named user and groups, allowing the actual reported to the client-side API a matter of client choice, as it should be.

F.3. Possible Extensions for Appending to Files

It is possible to incorporate such extensions in a new minor version or to an existing extensible minor version (e.g., NFSv4.2) as provided for in [RFC8178]. There are a number of possibilities to be considered with regard to extensions of OPEN and provisions to be made for effecting each subsequent extension

We will deal first with possibilities for the extension of the OPEN operation through the addition of one or more new access flags to the OPEN operation. As required by [RFC8178]. there needs to be way for the client to determine when server support is present. We are assuming that the approach specified in [RFC9754] will be available. The following possibilities need to be considered:

* The provision of a new flag, which, when set allows the resulting open file to only be appended to and when not set only allow WRITEs that do not change the EOF.

  This approach, while changing the semantics of WRITE, does not require any extension to WRITE or the creation of a new APPEND. The nature of the operation would be determined based on the characteristics of the stateid passed to WRITE.

* The provision of two new flags separately allowing WRITEs which append to the file and WRITEs which modify existing bytes and do not extend the file. In this case, WRITEs which both modify existing bytes and extend EOF are hard to deal with since it they require the choice to be made at WRITE time and are at variance with the expectation that this choice be made at OPEN time.

This would require some means of distinguishing these separate operations when the IO was requested. This might involve a special reserved parameter to WRITE or the creation of a new APPEND operation.

* The provision of a new flag, which, when set allows the resulting open file to be appended to using special append-only operations which might be effected, as above by a WRITE with special reserved parameters or with a new APPEND operation

  While this would make the handling of append-only files more convenient, it would not provide any help in making the appropriate ACE mask checks, since a non-append write would need to be checked if to see if it was overwriting existing bytes.

In the final two cases discussed above, it is necessary to signal at the time of the IO request whether appending or writing is to be done. Because it is difficult to add a flag to WRITE, this would require a special argument (e.g. offset of 2^64-1) or the definition of a new APPEND operation.

F.4. Possible Extensions to Support Massive ACLs

[Consensus Needed (Item #120f), Through end of section]: Even when filesystems provides adequate space for very large ACLs, it is possible that there may arise a need for ACLs so large that the request to set them or the response needed to interrogate them, may be larger than what can be provided than typical session implementations.

This would make it necessary provide extensions that support such massive ACLs. When designing such facilities, the following design considerations should be kept in mind.

* Although providing new operations GetBigAcl an SetBigAcl would address the issue, it would be better we provided A common facility to support multiple possible large attributes including possible support for large instances of draft-POSIX ACLs described in Appendix E.

* While transferring the attribute to/from a file is a good way to do this, we want to avoid unnecessary work with directory management and IOs to persistent storage, if they can be avoided.

* We want to avoid the client having to do cleanup to avoid the accumulation of ACL manipulation detritus.

A likely approach that should be considered would allow the opening of unnamed temporary file that looks, to the server and client, files all of whose link have been removed and will be silently deleted when closed, if not referenced from elsewhere before that close.

If facilities to create these are present, then:

* A GETATTR_PLUS operation could accept an array of attribute numbers, with the value of each such attribute returned in the form of the stateid of an unnamed temporary file opened for READ together with an associated filehandle,

  The values of the large attributes could be obtained using READ with the file going away when closed.

* A SETATTR_PLUS operation could accept an array of attribute numbers, with a corresponding array in which the value of the attribute to be set was expressed as the stateid of an unnamed temporary file.

  The large values to be presented could be stored in unnamed temporary files opened for WRITE, and presented to SETATTR_PLUS. Once this is done, those files could be closed.

Acknowledgments

The author wishes to thank Tom Haynes for his helpful suggestion to deal with security for all NFSv4 minor versions in the same document. The benefits of this approach have been even more apparent in dealing with the issues associated with ACL description.

The author wishes to thank Bruce Fields for his helpful comments regarding ACL support which had a major role in the evolution of this document.

The author wishes to thank Rick Macklem, Trond Myklebust, and Chuck Lever for their help in clarifying and addressing issues relating to POSIX ACL support.

The author wishes to thank Rick Macklem for his helpful work to deal with issues related to draft-POSIX ACLs with respect to both this document and others.

The author wishes to thank Rick Macklem and Pali Rohar for their invaluable help in dealing with issues related to the interaction of Windows semantics and NFSv4 ACLs.

RFC Editor Notes

[RFC Editor: please remove this section prior to publishing this document as an RFC]

[RFC Editor: prior to publishing this document as an RFC, please replace all occurrences of RFCTBD10 with RFCxxxx where xxxx is the RFC number of this document]

[RFC Editor: prior to publishing this document as an RFC, please replace all occurrences of RFCTBD20 with RFCyyyy where yyyy is the RFC number of the document providing an overall description of NFSv4 security, currently expected to result from completion of the document referenced in [I-D.dnoveck-nfsv4-security] or a document replacing that one.

Author’s Address

David Noveck (editor)
NetApp
201 Jones Road, Suite 16
Waltham, MA 02451
United States of America
Phone: +1-781-572-8038
Email: davenoveck@gmail.com